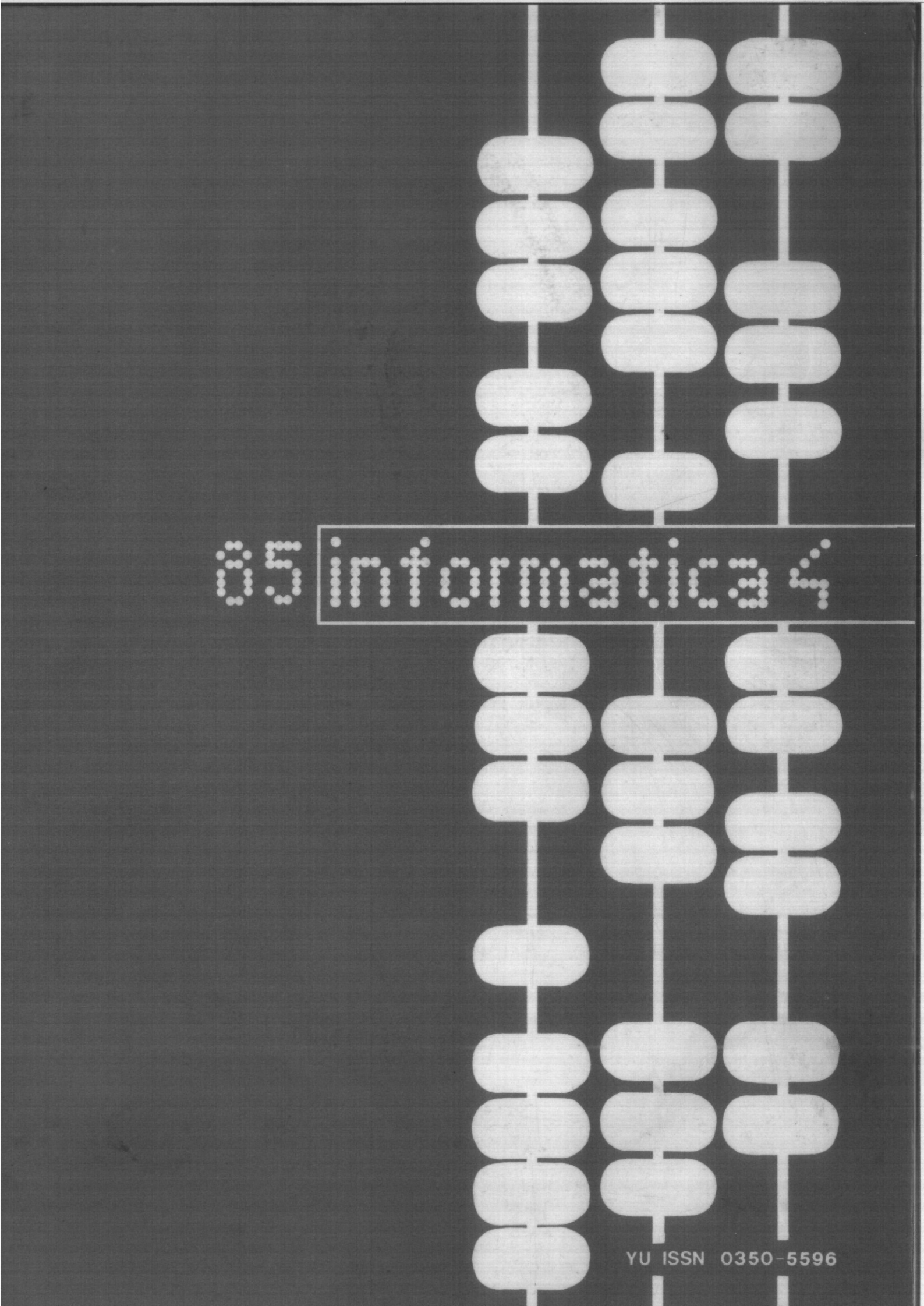


YU ISSN 0350-5596

informatica

ČASOPIS ZA TEHNOLOGIJO RAČUNALNIŠTVA
IN PROBLEME INFORMATIKE
ČASOPIS ZA RAČUNARSKU TEHNOLOGIJU I
PROBLEME INFORMATIKE
SPISANIE ZA TEHNOLOGIJA NA SMETANJETO
I PROBLEMI OD OBLASTA NA INFORMATIKATA

YU ISSN 0350–5596

LETNIK 9, 1985 – Št. 4

Časopis izdaja Slovensko društvo INFORMATIKA, 61000 Ljubljana, Parmova 41, Jugoslavija

UREDNIŠTVO IN UPRAVA: Informatica, Parmova 41, 61000 Ljubljana; telefon (061) 312-988; teleks 31366 YU Delta

LETNA NAROČNINA za delovne organizacije znaša 2900 din, za redne člane 790 din, za študente 290 din; cena posamezne številke je 890 din. ŽIRO RAČUN: 50101 - 678 - 51841.

Pri financiranju časopisa sodeluje Raziskovalna skupnost Slovenije.

Na podlagi mnenja Republiškega sekretariata za prosveto in kulturo št. 4210-44/79, z dne 1.2.1979, je časopis oproščen temeljnega davka od prometa proizvodov

TISK: Tiskarna Kresija, Ljubljana

GRAFIČNA OPREMA: Rasto Kirn

18. jugoslovansko mednarodno posvetovanje za računalniško tehnologijo in uporabo
Nova Gorica, 24.–26. september 1985

ZBORNIK DEL

INFORMATICA 85

Uredil: Marko Rogač

Programski odbor: Anton P. Železnikar
Janez Grad
Nada Lavrač
Stanko Cufer
Marko Rogač

Prispevke objavljamo v prvotni obliki. Na osnovi ocen recenzentov je programski odbor uvrstil predložene prispevke v program "Informatica 85" in jih razvrstil v deset tehničnih področij.

informatica

JOURNAL OF COMPUTING AND INFORMATICS

YU ISSN 0350–5596

VOLUME 9, 1985 – No. 4

Published by INFORMATIKA, Slovene Society for Informatics, Parmova 41, 61000 Ljubljana, Yugoslavia

HEADQUARTERS: Informatica, Parmova 41, 61000 Ljubljana, Yugoslavia
Phone: 61-312-988; Telex: 31366 YU DELTA

ANNUAL SUBSCRIPTION RATE: US$ 22 for companies, and US$ 10 for individuals

PRINTED BY: Tiskarna Kresija, Ljubljana

DESIGN: Rasto Kirn

18th Yugoslav International Conference on Computer Technology and Usage
Nova Gorica, September 24-26, 1985

PROCEEDINGS

INFORMATICA 85

Edited by Marko Rogač

Programe Committee: Anton P. Železnikar
Janez Grad
Nada Lavrač
Stanko Cufer
Marko Rogač

The contributed texts are in their submitted form. The reports reviewed extended summaries of the communications. The communications are contained in the programe "Informatica 85" and classified in teh technical areas.

V S E B I N A

UMETNA INTELIGENCA

PETA RAČUNALNIŠKA GENERACIJA

CONTENTS

PLENARNA ZASEDANJA

PLENAR MEETING

INTERACTIVE SYSTEM PERFORMANCE MEASUREMENT AND ANALYSIS

Jozo J. Dujmovic
Department of Electrical Engineering
University of Belgrade, Yugoslavia

UDK: 681.3.02

ABSTRACT: Two techniques for benchmarking interactive systems are proposed in the paper. The techniques are based on synthetic benchmark programs which simulate two typical workloads: (1) a transaction processing workload, and (2) a program development workload. These workloads reflect two fundamental activities that are regularly encountered in all interactive systems. In the case of the transaction processing workload the proposed benchmark programs are near machine-independent and include a synchronization mechanism enabling simultaneous start of a number of independent processes, one process per each interactive terminal, as well as the automatic collection and processing of measured performance indicators. In the case of the program development workload the proposed benchmark programs are shorter and simpler than in the case of the transaction processing workload, but are machine-dependent and require more operator activities. In both cases a single evaluator can realize the complete performance measurement and then use analytic models to extend the measured results. The proposed performance measurement approach is verified in a VAX/VMS environment.

1. INTRODUCTION

The basic goal of benchmarking interactive systems is to evaluate the performance of multi-access computer systems serving a number of interactive users. Interactive performance measurements are generally considered complex and expensive [BEN75]. They frequently include specialized minicomputers for simulating the activity of terminals and for generating an interactive workload, as well as for collecting and processing of measured results [SPO79]. Benchmarking of interactive systems sometimes requires many machine-dependent activities and/or the use of a number of specially trained human operators for executing an interactive workload according to a given script [BEN75]. Obviously, such activities are both complex and expensive.

In this paper our main goal is to propose an interactive system performance measurement technique that is relatively simple and inexpensive. The most convenient performance analysis technique depends on the purpose of such an analysis and in this paper we are primarily interested in benchmarking techniques suitable for computer evaluation and selection processes. So, we need a performance measurement technique that can be applied both for the efficient performance measurement of real interactive systems and for the processing of measured results. To that end the following requirements should be satisfied:

1. Performance measurements must be as simple, realistic, and inexpensive as possible.

2. To achieve the simplicity of measurement benchmark programs should be relatively short, and, in all cases where it is possible, machine-independent.

3. To provide a realistic environment and reliable results it is necessary to organize measurements using real terminal equipment connected to real communication and control units.

4. To minimize the cost of measurement it is necessary that a single evaluator can perform all measurement-related manipulation with a set of terminals, and the measurement time should be kept within reasonable limits.

In this paper we propose two different benchmarks for interactive system performance measurement and analysis: (1) Interactive Transaction Processing Benchmark (ITPB) and (2) Interactive Program Development Benchmark (IPDB). Both benchmarks completely satisfy the above requirements but they fundamentally differ in the workload they use. ITPB is based on a simple synthetic workload consisting of a repeating cycle which includes a think time period followed by a sequence of disk accesses and CPU activities. Such a synthetic workload simulates a typical transaction processing where each interactive terminal is used for performing essentially the same task. As opposed to that, IPDB is based on a natural workload and performs a typical interactive program development cycle including editing, listing, compilation, linking, and execution of selected real programs. Therefore, IPDB uses various systems programs including compilers and linkers and integrally measures interactive performance of a complete computer system with all of its hardware and software components. Such an approach eventually yields a different workload for each of competitive computers due to different resource consumption of competitive systems programs. Of course, this type of different workloads is completely justifiable since it exactly reflects real life situations where each computer's global efficiency significantly depends on the efficiency of systems programs. On the other hand, ITPB generates the same synthetic workload for all competitors.

2. AN INTERACTIVE TRANSACTION PROCESSING BENCHMARK (ITPB)

Interactive transaction processing is a frequent interactive application in banking, reservation and sale of tickets, inventory control, and in many other areas of data processing for business systems, information retrieval systems, etc. An interactive computer used for transaction processing is assumed to serve a number of interactive terminals. Each interactive terminal processes a relatively simple workload. In a typical situation a terminal operator is serving customers and at the beginning of service the operator accepts customer's service request and then uses his/her terminal to type-in a processing request. The interval from the beginning of service to the end of typing is called a "think time" and it may vary from few seconds to few minutes depending on the type of service and the frequency of customers. When the processing request is submitted for processing the computer typically performs a sequence of disk accesses and processor activities, and then displays (or prints) a number of lines of an answer to the operator's request. Te same routine repeats with other customers. Therefoe, the above interactive workload can be simulated as follows:

```
for k := 1 to TRANSACTIONS do
    begin
       Wait during the think time interval;
       for i:= 1 to CYCLES do
           begin
              Perform NACCESS[i] disk accesses ;
              Perform processor activity
                 specified by MILLISEC[i] ;
              Display LINES[i] text lines
           end
    end
```

In this algorithm NACCESS[i], MILLISEC[i], and LINES[i] are respectively the number of disk accesses, a processor activity parameter, and the number of displayed lines in i-th cycle. If THINKTIME denotes a constant average value of the think time interval then a spectrum of interactive workloads can be precisely specified using the following parameters: TRANSACTIONS, THINKTIME, CYCLES, NACCESS[1..CYCLES], MILLISEC[1..CYCLES], and LINES[1..CYCLES]. A FORTRAN implementation of the above interactive workload can be organized as shown in Listing 1.

The primary goal of benchmarking interactive systems is to determine actual changes of performance indicators caused by increasing the number of active terminals. If the total number of available terminals is MAXTERM then we should measure the performance indicators of a computer serving 1, 2, ..., MAXTERM terminals. The simplest way to perform MAXTERM independent measurements with an increasing number of active terminals is to organize a completely automatic measurement procedure with MAXTERM copies of an interactive benchmark program (INTERBEN) and a control program (CTRL) which successively starts the groups of 1, 2, ..., MAXTERM benchmark programs using the technique of process sinchronization with global semaphores. During the performance measurement INTERBEN programs save all measured performance indicators in a shared file (TIMES.DAT) and at the end of measurement the CTRL program reads the accumulated results and performs a final processing which includes both the computation of all important global performance indicators and the prediction of system performance for cases where more than MAXTERM terminals are active. The organization of CTRL and INTERBEN programs is presented in Fig. 1, and all components of ITPB are shown in Fig. 2. A FORTRAN implementation of two basic programs CTRL and INTERBEN is presented in Listing 2. Results of a typical performance measurement using ITPB are shown in Fig. 3.

3. AN INTERACTIVE PROGRAM DEVELOPMENT BENCHMARK (IPDB)

Interactive development of new programming systems is a permanent activity in almost all computing centers, and in computing centers serving universities and research organizations program development may be a predominant activity. During the program development typical interactive users generate a workload characterized by relatively long think times and frequent activating of various systems programs including oparating system, editors, compilers, linkers, debuggers, etc. So, in many cases it may be rather important to measure the performance of competitive computers processing an interactive program development workload.

Let us first clearly specify the essential property of IPDB: it mainly measures the efficiency of systems programs and consequently the workload of competitive computers must differ. In a general case the efficiency of relevant systems programs indirectly affects the efficiency of all benchmark programs: for example, the processing speed of FORTRAN and COBOL benchmark programs partially depends on various optimizing properties of pertinent FORTRAN and COBOL compilers. However, the activity of compilers precedes all general batch measurements, and in such cases the computer workload is essentially the same for all competitive systems. As opposed to that during the benchmarking based on IPDB we usually wish to explicitely measure the speed of compilation and linking, and obviously each computer uses its own compiler and linker yielding inevitably defferent workloads. Of course, such measurements are both justifiable and important because they realize exactly those processing activities that really occur in practice. The only disadvantage of such an approach is that albeit it can be rather general it cannot be machine-independent. Hence our pesentation of IPDB must be based on a real operating system and we will exemplify IPDB using VAX/VMS.

A typical program development process includes one or more of the following activities: (1) display of source programs, (2) editing of source programs, (3) compilation, (4) linking, and (5) execution of linked programs. Each of these activities is performed interactively and it consists of two basic intervals called "think time" and "run time". During the think time interval the terminal user is assumed to analyze previous results, to prepare a new task, and to type a corresponding service request. During the run time interval the user is waiting for results of the previously submitted request and then the same interaction cycle repeats. The sum of the think time and the run time is termed the interaction cycle time, and one of main tasks of benchmarking interactive systems is to measure the interaction cycle times as a function of the number of active terminals.

According to the above presentation it is convenient to organize an interactive benchmark program using for each program development interaction a separate "interactive block" presented in Fig. 4. We assume that a fixed number of blocks is to be executed during each

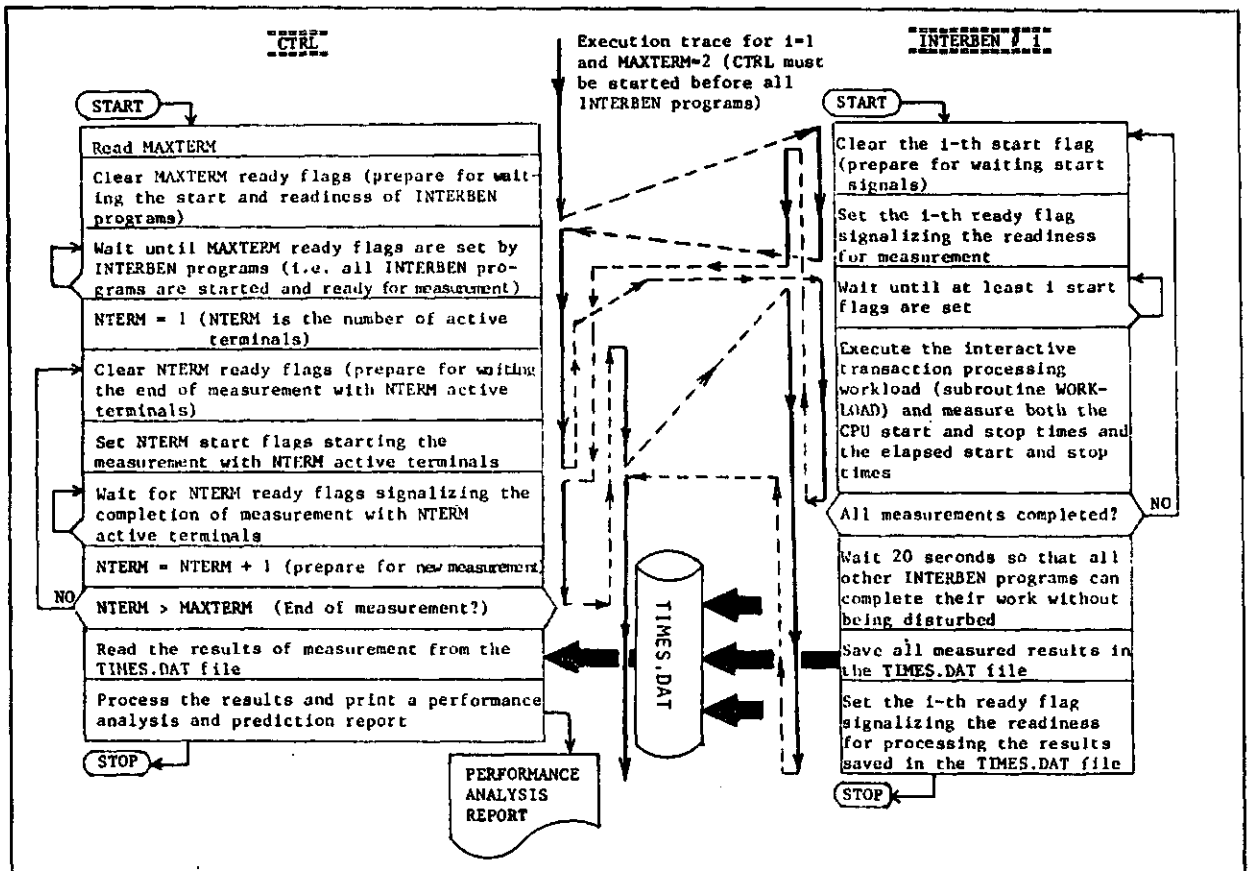


Figure 1. The structure and synchronization of CTRL and INTERBEN programs

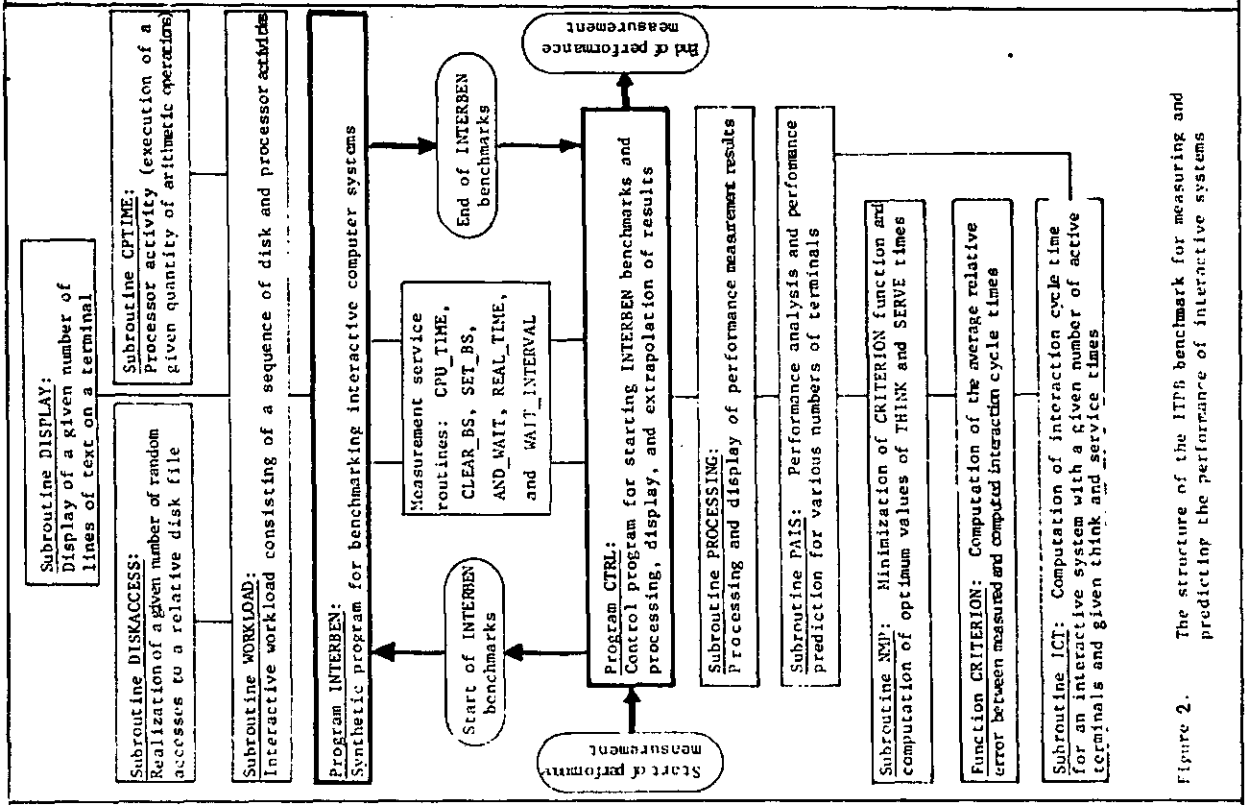


Figure 2. The structure of the ITPB benchmark for measuring and predicting the performance of interactive systems

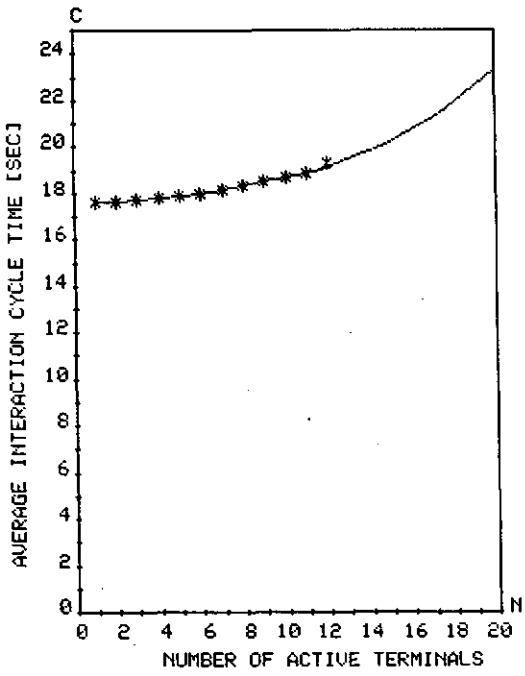


Figure 3

measurement and consequently each block must contain an end-of-measurement test comparing a current value of a block counter with a given maximum number of blocks to be executed. An IPDB consists of a cyclic chain of interactive blocks where each block performs a selected program development task. Fig. 5 shows such a chain reflecting the following scenario of interactive program development:

Block #1 (LIST)

The interactive user displays a source program and discovers a need to modify the program.

Block #2 (EDIT)

Editing process which creates a new source file.

Block #3 (COMPILE)

Compilation process generating a file with compilation messages and, if no errors are detected, a corresponding object file.

Block #4 (LIST)

Display of the file containing the compilation messages.

Block #5 (EDIT)

Assuming an error is encountered or an additional improvement is needed a new editing process is performed.

Block #6 (COMPILE)

Compilation of corrected source programs generating both list and object files.

Block #7 (LINK)

Linkage of all necessary object files creating an executable program file.

Block #8 (EXECUTE)

Execution of the linked program.

The above program development cycle is rather typical but it can also be easily modified according to user's needs. The proposed interactive blocks can be easily replaced by some other blocks, and both their number and their ordering can be changed. Once we have a cyclic chain of interactive blocks we are faced with the following problem: if all termnals are used to simultaneously start the same sequence of blocks then an artificial situation is created where all hypothetical terminal users simultaneously list their files, simultaneously think, compile link, etc., causing unnaturally concentrated excessive requests for system resources. So, it is necessary to uniformly distribute various program development activities of supposedly unrelated terminal users and that can be done as proposed in Fig. 5 where each terminal user has its own entry point in the cyclic structure of IPDB. To realize such a benchmark program with multiple entry points, some interactive blocks are to be skipped yielding a corresponding block select test at the beginning of each block as shown in Fig. 4. A program realization of IPDB using DCL for VAX/VMS is presented in Listing 3. The technique of starting the measurement now differs from the control program technique used in the case of ITPB. To simplify the IPDB the system time is used for synchronizing the start of all simultaneous copies of IPDB. Results of a typical performance measurement using IPDB are shown in Fig. 6.

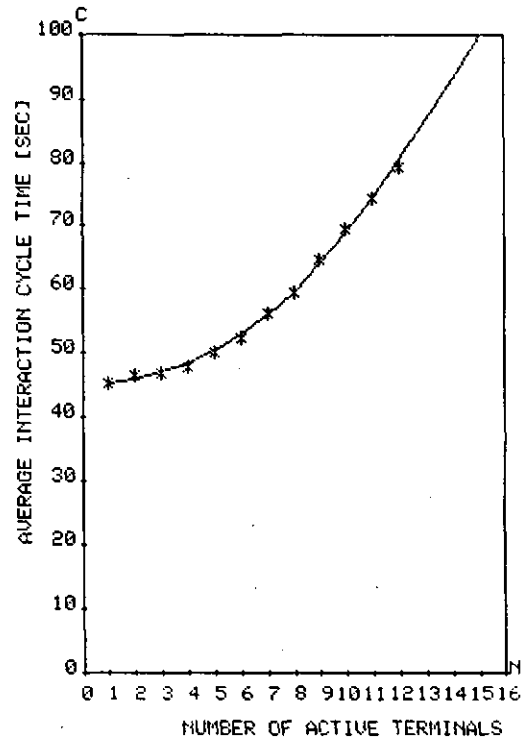


Figure 6

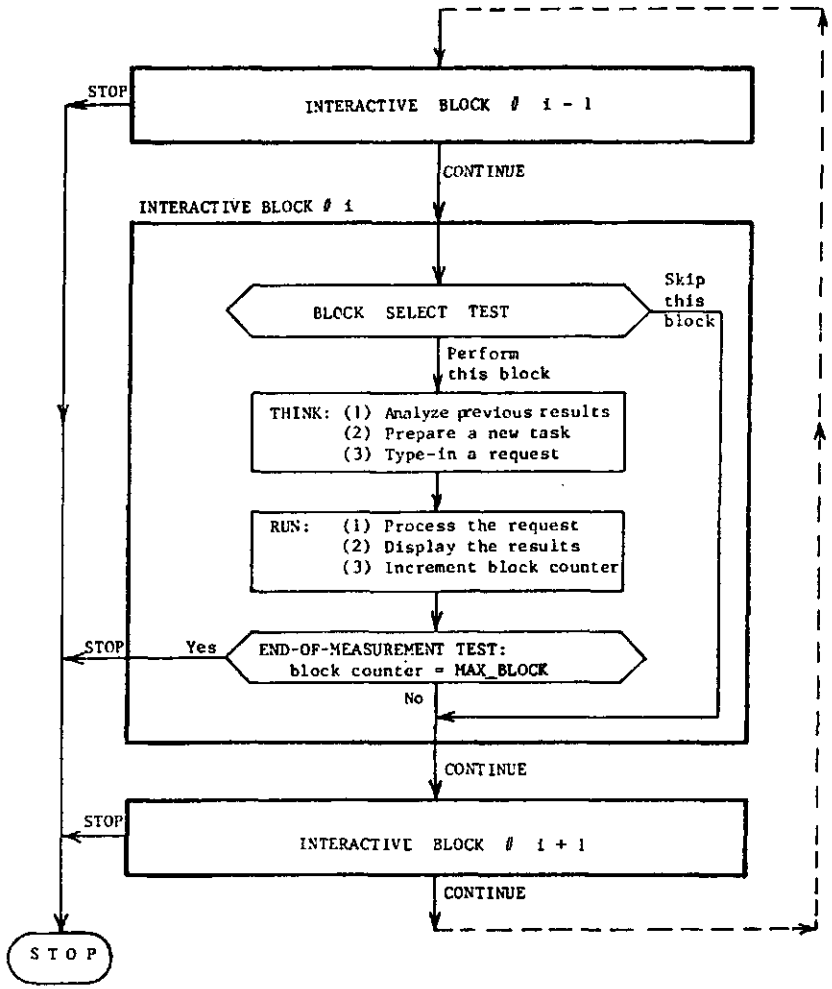


Figure 4. Basic structure of an interactive block

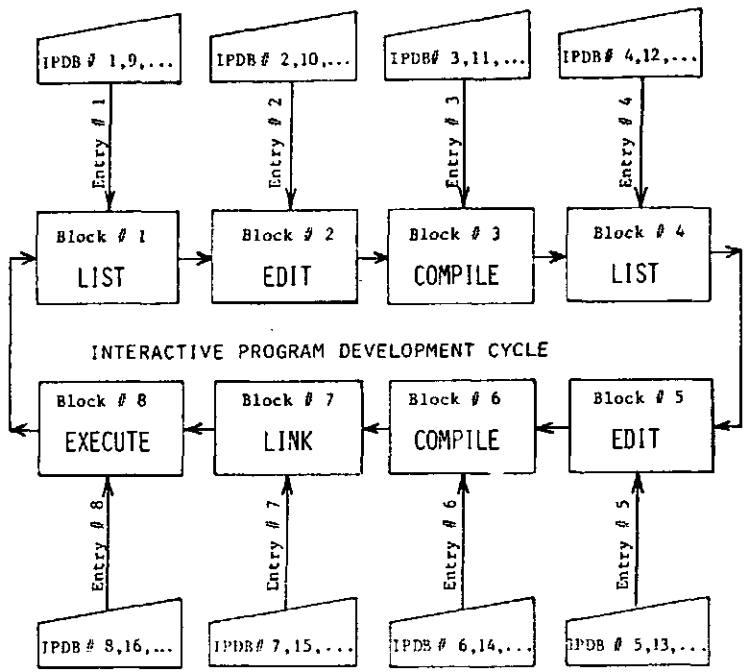


Figure 5. An interactive program development cycle with a uniform distribution of entry points for IPDB workloads

```
C=====================================================================
C     INTERACTIVE TRANSACTION PROCESSING WORKLOAD
C=====================================================================
      SUBROUTINE WORKLOAD (ID)          ! ID = workload identification
      INTEGER      TRANSACTIONS, CYCLES
      PARAMETER    (TRANSACTIONS = 50, THINKTIME = 15., CYCLES = 5)
      DIMENSION    NACCESS(CYCLES),MILLISEC(CYCLES),LINES(CYCLES)
      CHARACTER*80 TEXT
      COMMON       MOD,MUL,KEY,TEXT
C---- W O R K L O A D   P A R A M E T E R S ----------------------+
      DATA NACCESS  /8,5,5,8,4/          ! Number of disk accesses  !
      DATA MILLISEC /40,30,50,30,50/     ! Processing times         !
      DATA LINES    /2,3,2,3,2/          ! Number of displayed lines !
      MOD  = 99991                       ! Initialization of random  !
      MUL  = 9973                        ! number generator: modulus,!
      KEY  = ID                          ! multiplier & initial value!
      TEXT = 'initial string'            !---------------------------+

      DO K   = 1,TRANSACTIONS
         KEY = KEY*MUL - KEY*MUL/MOD*MOD     ! Random KEY generation
         URN = FLOAT(KEY)/MOD                ! 0 < URN < 1 (uniform
         CALL WAIT_INTERVAL (THINKTIME*URN)  ! random number)
         DO I = 1,CYCLES
            IF (NACCESS(I)  .GT. 0)  CALL DISKACCESS (NACCESS(I))
            IF (MILLISEC(I) .GT. 0)  CALL CPTIME     (MILLISEC(I))
            IF (LINES(I)    .GT. 0)  CALL DISPLAY    (LINES(I))
         END DO
         PRINT *,'>>>>----------------------> Transaction #', K
         CALL WAIT_INTERVAL (THINKTIME*(1.-URN))
      END DO
      RETURN
      END
C=====================================================================
C     DISK ACCESS SUBROUTINE
C=====================================================================
      SUBROUTINE DISKACCESS (NACCESS)      ! NACCESS = number of
      COMMON MOD,MUL,KEY,TEXT              ! random accesses
      CHARACTER*80 TEXT
      DO  I = 1,NACCESS
          KEY = KEY*MUL - KEY*MUL/MOD*MOD  ! Random KEY generation
          READ (UNIT=11,FMT=10,REC=KEY) TEXT
   10     FORMAT(A)
      END DO
      RETURN
      END
C=====================================================================
C     PROCESSOR TIME SUBROUTINE
C=====================================================================
      SUBROUTINE CPTIME (MILLISEC)      ! MILLISEC = processor
      PARAMETER (LOOP = 216)            !        activity parameter
      LTEMP = MILLISEC                  ! LOOP = program parameter
      KTEMP = MILLISEC                  !        to be adjusted so
      DO  J = 1,MILLISEC                !        that
          DO  I = 1,LOOP                !        CPUTIME(1) = 1 ms
              LTEMP = LTEMP-I           !        for a selected
              KTEMP = KTEMP+I           !        computer
          END DO
          LTEMP = (LTEMP+KTEMP)/2       ! i.e.  LTEMP = MILLISEC
          KTEMP = LTEMP
      END DO
      MILLISEC = KTEMP                  ! i.e.  MILLISEC remains
      RETURN                            !       unchanged
      END
C=====================================================================
C     DATA DISPLAY SUBROUTINE
C=====================================================================
      SUBROUTINE DISPLAY (LINES)        ! LINES = the number of
      CHARACTER*80 TEXT                 !         80-character lines
      COMMON MOD,MUL,KEY,TEXT           !         to be displayed
      DO L = 1,LINES
         PRINT *, TEXT
      END DO
      RETURN
      END
C---------------------------------------------------------------------
```

Listing 1

```
C=====================================================================
C     CTRL - Control program for benchmarking interactive systems
C            (This program must precede all INTERBEN programs!
C=====================================================================
      INTEGER   CTRL
      COMMON MAXTERM          ! MAXTERM = total number of terminals used for
      PARAMETER (INTERBEN=1, CTRL=2)              ! performance measurement
      OPEN (UNIT=22,FILE='TIMES.DAT',STATUS='NEW',ACCESS='DIRECT',
     ,     FORM='FORMATTED',RECL=80,INITIALSIZE=35 ! Creation of TIMES.DAT
     ,     ,ORGANIZATION='RELATIVE',SHARED)       ! (For 20 terminals there
      CLOSE (UNIT=22)                             ! are 210 records. If
      PRINT*,' ENTER THE TOTAL NUMBER OF TERMINALS'! block contains 6 records
      READ *, MAXTERM                             ! then size = 35 blocks.)

      DO NTERM = 1,MAXTERM                  !-----------------------------+
         CALL CLEAR_BS (CTRL,NTERM)         ! Wait until all INTERBEN     !
      END DO                                ! programs are started        !
      CALL AND_WAIT (CTRL,2**MAXTERM-1)     ! and ready for measurement   !
                                            !-----------------------------+
      DO NTERM = 1,MAXTERM                  ! Main measurement loop       !
         DO I = 1,NTERM                     ! Activate NTERM programs     !
            CALL CLEAR_BS (CTRL,I)          ! Preparation for wait        !
            CALL SET_BS   (INTERBEN,I)      ! Start of the I-th INTERBEN  !
         END DO                             !-----------------------------+
         CALL AND_WAIT (CTRL,2**NTERM-1)    ! Wait for completion of      !
      END DO                                ! INTERBEN programs           !
                                            !-----------------------------+
      CALL PROCESSING                       ! Processing and display of   !
      STOP                                  ! all measured results        !
      END                                   !-----------------------------+
C=====================================================================
C     INTERBEN - Synthetic program for benchmarking interactive systems
C          (This program must be started after the control program CTRL)
C=====================================================================
      INTEGER   CTRL
      DIMENSION START_CPU_TIME(20), STOP_CPU_TIME(20)
      REAL*8    START_TIME(20), STOP_TIME(20), REAL_TIME
      PARAMETER (INTERBEN=1, CTRL=2)
      PRINT *, ' ENTER THE TOTAL NUMBER OF TERMINALS ',
     ,         'AND THE TERMINAL IDENTIFICATION NUMBER'
      READ  *, MAXTERM, ID
      OPEN (UNIT=11,FILE='DIRREL.DAT',STATUS='OLD',ACCESS='DIRECT',
     ,     FORM='FORMATTED',SHARED)
C-------------------------------------------------+---------------------!
      DO NTERM = ID,MAXTERM                     ! NTERM=Total number of   !
C                                               !       active terminals  !
         CALL CLEAR_BS (INTERBEN,ID)            ! Preparation for wait    !
         CALL SET_BS   (CTRL,ID)                ! Ready for measurement   !
         CALL AND_WAIT (INTERBEN,2**NTERM-1)    ! Wait for NTERM-start    !
C                                               ! flags                   !
C--------- M e a s u r e m e n t ---------------+-------------------------+
         START_TIME (NTERM)     = REAL_TIME ( ) ! Each WORKLOAD performs  !
         START_CPU_TIME (NTERM) = CPU_TIME  ( ) ! the same amount of CPU  !
            CALL WORKLOAD (ID)                  ! and disk operations.    !
         STOP_TIME (NTERM)      = REAL_TIME ( ) ! ID is used as a seed    !
         STOP_CPU_TIME( NTERM)  = CPU_TIME  ( ) ! of random # generator.  !
C-------------------------------------------------+-------------------------+
      END DO
      CALL WAIT_INTERVAL (20.)                  ! Wait 20 seconds
      CLOSE (UNIT=11)
      OPEN (UNIT=22,FILE='TIMES.DAT',STATUS='OLD',ACCESS='DIRECT',
     ,     FORM='FORMATTED',SHARED)
      DO NTERM = ID,MAXTERM                     ! Save all results
         WRITE (UNIT=22, REC=ID+NTERM*(NTERM-1)/2, FMT=20)
     ,        START_TIME(NTERM), START_CPU_TIME(NTERM),
     ,        STOP_TIME(NTERM) , STOP_CPU_TIME(NTERM)
      END DO
   20 FORMAT (4E20.12)
      CLOSE (UNIT=22)                           !-------------------------+
      CALL SET_BS (CTRL,ID)                     ! End of measurement and  !
      STOP                                      ! start of processing     !
      END                                       !-------------------------+
```

Listing 2

```
$!==========================================================================
$!       IPDB  -  INTERACTIVE PROGRAM DEVELOPMENT BENCHMARK
$!==========================================================================
$ OPEN/READ INFILE START.DAT ! START.DAT contains the start of measurement time
$ READ INFILE STARTTIME      ! Read STARTTIME as a character string (HH:MM:SS)
$ CLOSE INFILE               ! INFILE is a logical name assigned to START.DAT
$!--------------------------------------------------------------------------
$ MAX_BLOCK  == 24                  ! Requested number of processed blocks
$ NBL        = 8                    ! NBL := the number of blocks in IPDB
$ ID         == 1                   ! ID = terminal id. number (1,2,...)
$ FLAG       =  ID-1 - (ID-1)/NBLAMBL ! 0 <= FLAG <= NBL-1
$ START_TIME == STARTTIME           ! Define START_TIME as a global symbol
$ COUNT      == 0                   ! COUNT := the number of processed blocks
$ @ WAITSTART                       ! Wait until time-of-day = START_TIME
$!--------------------------------------------------------------------------
$ BLOCK_1:                          !       L I S T
$    IF FLAG .GE. 1 THEN GOTO BLOCK_2 !=====================================
$       @ THINK 15                  ! Wait 15 seconds (think and typing time)
$       TYPE MAIN1.FOR, PSCI1.FOR   ! List source programs
$       @ STOPTEST                  ! COUNT := COUNT+1;
$                                   ! if COUNT = MAX_BLOCK then STOP
$!--------------------------------------------------------------------------
$ BLOCK_2:                          !       E D I T
$    IF FLAG .GE. 2 THEN GOTO BLOCK_3 !=====================================
$       @ THINK 15                  ! Wait 15 seconds (think and typing time)
$       RUN EDITING1                ! Simulation of file editing
$       @ STOPTEST                  ! Increment COUNT and test
$!--------------------------------------------------------------------------
$ BLOCK_3:                          !       C O M P I L E
$    IF FLAG .GE. 3 THEN GOTO BLOCK_4 !=====================================
$       @ THINK 15                  ! Wait 15 seconds (think and typing time)
$       FOR/LIST MAIN1, MINV1, PSCI1 ! Compile FORTRAN programs
$       @ STOPTEST                  ! Increment COUNT and test
$!--------------------------------------------------------------------------
$ BLOCK_4:                          !       L I S T
$    IF FLAG .GE. 4 THEN GOTO BLOCK_5 !=====================================
$       @ THINK 30                  ! Wait 30 seconds (think and typing time)
$       TYPE MAIN1, MINV1, PSCI1    ! List the obtained compilation reports
$       @ STOPTEST                  ! Increment COUNT and test
$!--------------------------------------------------------------------------
$ BLOCK_5:                          !       E D I T
$    IF FLAG .GE. 5 THEN GOTO BLOCK_6 !=====================================
$       @ THINK 15                  ! Wait 15 seconds (think and typing time)
$       RUN EDITING1                ! Simulation of file editing
$       @ STOPTEST                  ! Increment COUNT and test
$!--------------------------------------------------------------------------
$ BLOCK_6:                          !       C O M P I L E
$    IF FLAG .GE. 6 THEN GOTO BLOCK_7 !=====================================
$       @ THINK 15                  ! Wait 15 seconds (think and typing time)
$       FOR/LIST MAIN1, MINV1, PSCI1 ! Compile FORTRAN programs
$       @ STOPTEST                  ! Increment COUNT and test
$!--------------------------------------------------------------------------
$ BLOCK_7:                          !       L I N K
$    IF FLAG .GE. 7 THEN GOTO BLOCK_8 !=====================================
$       @ THINK 15                  ! Wait 15 seconds (think and typing time)
$       LINK MAIN1, MINV1, PSCI1    ! Link object programs
$       @ STOPTEST                  ! Increment COUNT and test
$!--------------------------------------------------------------------------
$ BLOCK_8:                          !       E X E C U T E
$       @ THINK 30                  !=======================================
$       RUN PSCI1                   ! Execution of linked program
$       @ STOPTEST                  ! Increment COUNT and test
$       FLAG = 0                    ! Continue without skiping blocks
$       GOTO BLOCK_1                ! Repeat the above sequence of blocks
$!--------------------------------------------------------------------------
```

```
$!==========================================================================
$!   STOPTEST PROCEDURE  (STOP WHEN THE NUMBER OF PROCESSED BLOCKS = MAX_BLOCK)
$!==========================================================================
$ COUNT == COUNT + 1                                  !  Increment block counter
$ WRITE SYS$OUTPUT ")))>----------------)  NUMBER OF EXECUTED BLOCKS = ', COUNT
$ IF COUNT .LT. MAX_BLOCK THEN EXIT                   !  Stop or continue ?
$ @ REALTIME                                          !  Measurement of real time
$ RUNTIME = ('F$EXTRACT(0,2,TIME_OF_DAY)'-'F$EXTRACT(0,2,START_TIME)')*3600  -
          + ('F$EXTRACT(3,2,TIME_OF_DAY)'-'F$EXTRACT(3,2,START_TIME)')*60    -
          + 'F$EXTRACT(6,2,TIME_OF_DAY)'-'F$EXTRACT(6,2,START_TIME)'
$ WRITE SYS$OUTPUT ' '
$ WRITE SYS$OUTPUT '*************************************************************'
$ WRITE SYS$OUTPUT '      IPDB SYSTEM  -  END OF PERFORMANCE MEASUREMENT'
$ WRITE SYS$OUTPUT '*************************************************************'
$ WRITE SYS$OUTPUT '      TERMINAL IDENTIFICATION NUMBER  = ', ID
$ WRITE SYS$OUTPUT '      TOTAL NUMBER OF EXECUTED BLOCKS = ', COUNT
$ WRITE SYS$OUTPUT '      TOTAL RUN TIME  = ', RUNTIME, '  SECONDS'
$ STOP                                !  >>>>>>>>>>>>  E n d   o f   m e a s u r e m e n t
$!--------------------------------------------------------------------------

$!==========================================================================
$!   WAITSTART PROCEDURE (WAIT A GIVEN TIME-OF-YEAR AND START THE MEASUREMENT)
$!==========================================================================
$ @ REALTIME                                          !  Measurement of real time
$ WAIT 'DATE''START_TIME' ! WAIT uses time-of-year format(DD-MMM-YYYY HH:MM:SS)
$ EXIT                    ! Return to a calling procedure
$!--------------------------------------------------------------------------

$!==========================================================================
$!   THINK PROCEDURE (WAIT A GIVEN TIME INTERVAL -  SIMULATION OF THINK TIME)
$!==========================================================================
$ WAIT 00:00:'P1'                     !  P1 is an input argument denoting seconds
$ EXIT                                !  Return to a calling procedure
$!--------------------------------------------------------------------------

$!==========================================================================
$!   REALTIME PROCEDURE  (MEASUREMENT OF DATE AND TIME)
$!==========================================================================
$ TIMESTRING  = F$TIME()      !  =  time-of-year string (DD-MMM-YYYY HH:MM:SS)
$ DATE        == F$EXTRACT(0,12,TIMESTRING)    !  DATE     = DD-MMM-YYYY string
$ TIME_OF_DAY == F$EXTRACT(12,8,TIMESTRING)    !  TIME_OF_DAY = HH:MM:SS string
$ EXIT                                         !  Return to a calling procedure
$!--------------------------------------------------------------------------
```

```
C==========================================================
C     EDITING  -  SIMULATION OF AN EDITING PROCESS
C==========================================================
      CHARACTER*80 TEXT
      COMMON MOD, MUL, KEY, TEXT
C---- W O R K L O A D    P A R A M E T E R S ----------------+
      DATA INTERACTIONS, THINKTIME  /6, 10./                 !
      DATA NACCESS, MILLISEC, LINES /20, 200, 10/            !
      MOD = 99991                          !  Initialization !
      MUL = 9973                           !  of random      !
      KEY = 1                              !  number generator !
C-----------------------------------------+-----------------+
      OPEN (UNIT = 11, FILE = 'DEREL.DAY', STATUS = 'OLD',
     &      ACCESS = 'DIRECT', FORM = 'FORMATTED', SHARED)
C---- E D I T    C Y C L E ---------------+=================!
      DO IMI = 1,INTERACTIONS              ! Start of edit cycle !
         CALL DISKACCESS     (NACCESS)     ! OPEN-LOAD-SAVE-CLOSE !
         CALL DISPLAY        (LINES)       ! L I S T              !
         CALL WAIT_INTERVAL (THINKTIME)    ! T H I N K            !
         CALL CPTIME         (MILLISEC)    ! P R O C E S S        !
      END DO                               ! Repeat the edit cycle!
      CLOSE (UNIT = 11)                    !=====================!
      STOP
      END
```

Listing 3

4. ANALYTIC MODELING OF INTERACTIVE SYSTEMS

Analytic models are frequently used in conjunction with benchmark measurements in order to derive various synthetic performance indicators and to predict the performance of measured systems in cases where their parameters are changed after the completion of performance measurement [FER83]. Analytic models can include various levels of detail and the following three models, ordered according to an increasing level of detail, will be discussed below:

- bottleneck analysis model,
- machine repairman model, and
- balanced central server model.

The bottleneck analysis model [DEN78, LAZ84] is shown in Fig. 7. The model includes N interactive terminals and a central subsystem containing a bottleneck resource. The bottleneck resource is defined as a resource having the maximum VbSb product, where Vb denotes the number of visits to the bottleneck device during each interaction and Sb denotes the corresponding average service time. Other indicators of the model include the average think time Z, the average response time R, the average interaction cycle time C:=Z+R, and the average throughput X. All terminals are assumed to be equal and to receive identical services from the computer system. Consequently, during each interaction all terminals must be served yielding the throughput

$$X = N/C = N/(Z+R) .$$

The throughput satisfies the relation

$$Xmin <= X <= Xmax$$

where Xmin corresponds to the case where N=1:

$$Xmin = 1/Cmin = 1/(Z+Rmin).$$

The minimum response time can be easily obtained as a net sum of all necessary service times without queueing delays:

$$Rmin = V1S1 + V2S2 + ... + VkSk ,$$

where k denotes the number of resources, Vi is the number of visits to the i-th resource per interaction, and Si is the i-th resource service time. If N>>1 then the bottleneck resource is saturated, limiting the global throughput of the system to X=Xmax. The throughput of the saturated bottleneck resource is Xb=1/Sb. In cases where the bottleneck resource is visited once during each interaction it obviously follows Xmax=Xb. Generally, if the bottleneck resource is visited Vb times per each interaction then

$$Xmax = Xb/Vb = 1/VbSb .$$

Therefore, the C(N) function has two asymptotes. The minimum load asymptote is a horizontal line

$$C = 1/Xmin = Z+Rmin$$

and the saturation asymptote is the rising line

$$C = N/Xmax = NVbSb .$$

The intersection of the asymptotes denotes the beginning of saturation and that important point defines the critical number of terminals Nc:

$$Nc = Xmax/Xmin = (Z+Rmin)/VbSb .$$

If N <= Nc then the operation of the system can be considered normal; however, if N > Nc then the system becomes saturated yielding poor response time defined by the formula

$$R = N/Xmax - Z = NVbSb - Z .$$

According to Little's formula we have that the number of jobs in the central subsystem (i.e. the average number of terminals waiting for system's response) is n = RX, and for a saturated system it follows

$$n = RXmax = N - ZXmax = N - Z/VbSb .$$

Obviously, for N >> 1 the ratio n/N approaches 1, i.e. the majority of terminals are in a nonproductive waiting state.

It is important to realize that Nc depends both on the workload and on various system parameters. Since system parameters are constant, and the workload can be easily changed, it follows that Nc primarily reflects the computer workload and can be easily adjusted to take any desired value. However, the interactive workload during benchmarking is approximately constant and consequently the corresponding Nc indicator of competitive systems can be successfully applied for their comparison.

It is useful to note that the above bottleneck analysis holds also for batch systems where Z=0 and C=R yielding

$$Xmin = 1/(V1S1 + ... + VkSk) ,$$

$$Xmax = 1/VbSb ,$$

$$Nc = (V1S1 + ... + VkSk)/VbSb ,$$

$$R = V1S1 + ... + VkSk , \quad N << Nc$$

$$= NVbSb , \quad N >> Nc$$

where Nc is now interpreted as the critical degree of multiprogramming.

To derive Nc from benchmark measurements we need a more detailed model of computer system than the bottleneck analysis model shown in Fig. 7. The first such a model to be considered here is the machine repairman model shown in Fig. 8 [SCH65, KLE68, KLE75]. The machine repairman model is obviously the simplest possible model of an interactive system since it represents the whole central subsystem by a single service center. Using this model Nc is defined as follows:

$$Nc = 1 + Z/Sp ,$$

i.e. during each think time interval the central subsystem is expected to serve all of the remaining Nc-1 terminals ((Nc-1)Sp = Z). The machine repairman model can be used very efficiently for modeling the interaction cycle time and for a precise computation of Nc. Let C(N,Nc) be the interaction cycle time computed from the machine repairman model and let Cm(N), N = 1, .. , Nmax be the set of measured interaction cycle times. Then the critical number of terminals can be determined by minimizing the dfference between measured and computed interaction cycle times:

$$D(y) := \frac{100}{Nmax} \sum_{N=1}^{Nmax} | Cm(N) - C(N,y) | / Cm(N)$$

$$D(Nc) = \min_{y>=1} D(y) .$$

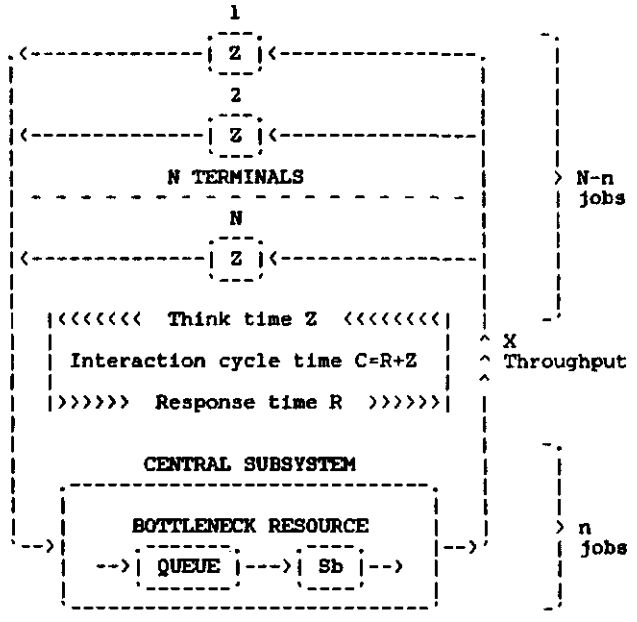


Figure 7. Bottleneck analysis model of an interactive system with N active terminals

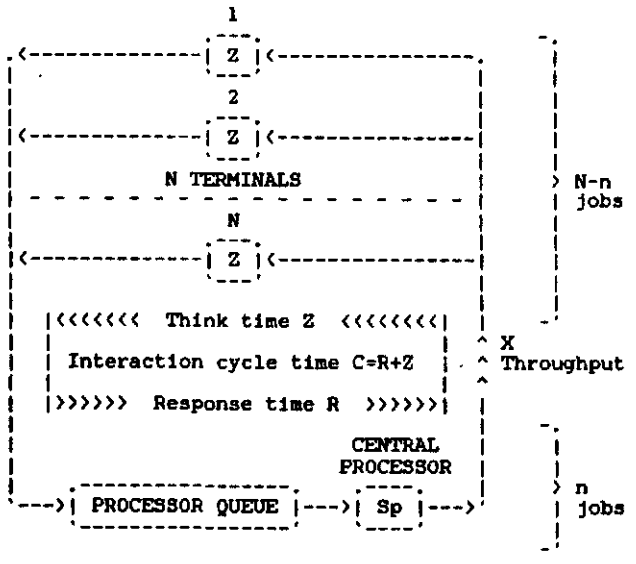
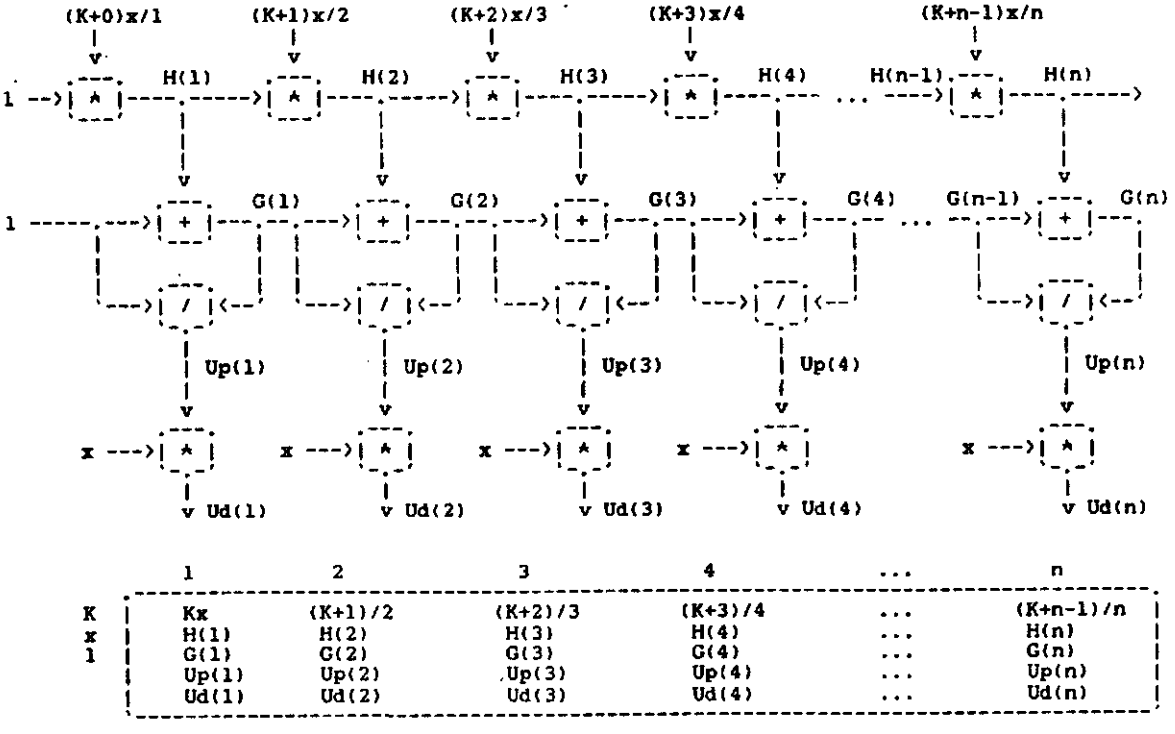


Figure 8. Machine repairman model of an interactive system



	1	2	3	4	...	n
K	Kx	(K+1)/2	(K+2)/3	(K+3)/4	...	(K+n-1)/n
x	H(1)	H(2)	H(3)	H(4)	...	H(n)
1	G(1)	G(2)	G(3)	G(4)	...	G(n)
	Up(1)	Up(2)	Up(3)	Up(4)	...	Up(n)
	Ud(1)	Ud(2)	Ud(3)	Ud(4)	...	Ud(n)

Figure 10. Numerical technique for direct computation of processor utilization and disk utilization for a balanced central server model of multiprogramming

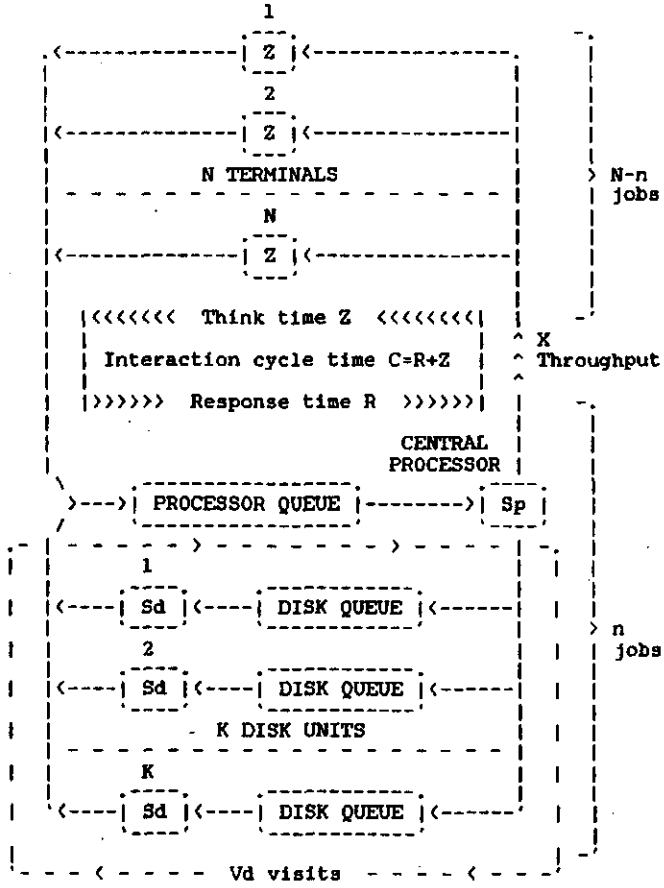


Figure 9. Balanced central server model and its parameters N, K, Vd, Z, Sp, and Sd

A FORTRAN program for computing Nc is presented in Listing 4. A sample output of this program exemplifying the analysis of an ITPB measurement of a VAX 11/750 is shown in Listing 5.

The machine repairman model cannot be used for detailed analyses of interactive systems with disk units. In such cases a more detailed analytic model is necessary and such a model is the central server model [BUZ71, BUZ73, CUR84]. In all performance measurements based on synthetic benchmark programs disk units are regularly uniformly utilized, i.e. the interactive workload is designed in such a way that each disk unit is accessed approximately the same number of times. The central server model in the case where the probability of accessing disks is equal for all units is shown in Fig. 9 and it is called the balanced central server model (BCSM). During each interaction each terminal is assumed to generate Vd disk accesses (i.e. Vd/K accesses per disk for a system with K disks units). In such a case the number of visits to central processor is Vp = 1+Vd per each interaction. After a visit to the central processor the interaction can be completed with the probability

$$p1 = 1/Vp = 1/(1+Vd)$$

or a given disk can be accessed with the probability

$$pd = (1-p1)/K = (1-1/Vp)/K = Vd/[K(1+Vd)] .$$

Let Sp be the average central processor service time a job receives during each visit to the central processor, and similarly, let Sd be the average disk service time. Assuming that the reader is familiar with Buzen's method for analyzing central server queueing networks [BUZ71, BUZ73] we present below only a modification of Buzen's algorithm that can be derived for the BCSM.

The probability of the state (n1,n2, ... , nk), (i.e. ni service requests at the i-th service center, i=1, ..., k, where the number of service centers is k = K+1) for the BCSM is

$$p(n1, n2, \dots, nk) = x^{n2 + \dots + nk} / G(n)$$

$$G(n) = \sum_{n1+\dots+nk = n} x^{n2 + \dots + nk} ,$$

$$x := pd\ Sd / Sp = Vd\ Sd / [K(1+Vd)Sp] .$$

The computation of the normalizing constant G(n) for the BCSM can be performed according to a simple algorithm which can be derived from the original Buzen's algorithm [BUZ73]. From the first row of Buzen's table it follows

$$G(1) = 1 + K\,x .$$

From the second row we have

$$G(2) = G(1) + K\,(K+1)\,x^2 / 2$$

and generally

$$G(i) = G(i-1) + \frac{K(K+1) \dots (K+i-1)}{i!} x^i$$

$$= G(i-1) + \binom{K+i-1}{i} x^i ,$$

$$i = 1,\dots,n , \qquad G(0) := 1 .$$

Therefore, the computation of the vector G for the BCSM can be done substantially simpler than in the original Buzen's algorithm. For n jobs in the central subsystem the processor utilization Up and the disk utilization Ud can be computed from

$$Up = G(n-1)/G(n) ,$$

$$Ud = x\ Up .$$

So, using the computational technique shown in Fig. 10 all fundamental parameters of the BCSM can be computed using the following simple algorithm which is suitable even for manual calculation using the table presented in Fig. 10:

```
G(0) := 1 ; H := 1 ; x := Vd*Sd/(Sp*K*(1+Vd));
FOR i := 1 TO n DO
    BEGIN
        H     :=  H*x*(K+i-1)/i ;
        G(i)  :=  G(i-1) + H ;
        Up(i) :=  G(i-1)/G(i) ;
        Ud(i) :=  x * Up(i)
    END
```

```
C=======================================================
C     MINIMIZATION OF F(X)                              |
C=======================================================
      SUBROUTINE FMIN1 (F, XSTEP, ERROR, XMIN, FMIN)
C-------------------------------------------------------+
C     F     = an external function to be minimized      |
C     XSTEP = initial value of step (constant or variable)|
C     ERROR = maximum absolute error of XMIN (const. or variable)|
C     XMIN  = an initial estimate of local minimum which is |
C             replaced by the resulting coordinate of minimum |
C     FMIN  = F(XMIN) = the resulting minimum of function F(X) |
C-------------------------------------------------------+
      STEP = XSTEP
      FMIN = F(XMIN)
      DO WHILE (ABS(STEP) .GE. ERROR * 0.1353)
         STEP  = - STEP
         XNEXT = XMIN + STEP
         FNEXT = F(XNEXT)
         DO WHILE (FNEXT .LT. FMIN)
            XMIN  = XNEXT
            FMIN  = FNEXT
            XNEXT = XNEXT + STEP
            FNEXT = F(XNEXT)
         END DO
         STEP = STEP / 2.7183
      END DO
      RETURN
      END

C=======================================================
C     RELATIVE MULTIPROGRAMMING ELAPSED TIME            |
C=======================================================
      FUNCTION RELTIME(NPROG,CRITPROG)
C-------------------------------------------------------+
C     NPROG    = number of active programs or terminals |
C     CRITPROG = critical number of active programs or  |
C                terminals (saturation point)           |
C     RELTIME  = relative run time:  RELTIME(1,C) := 1,  |
C                RELTIME(i,C) < RELTIME(i+1,C) , i >= 1 |
C-------------------------------------------------------+
      IF (CRITPROG .EQ. 1.) THEN
          RELTIME = NPROG
          RETURN
      END IF
      RHO  = 1. / (CRITPROG - 1.)
      SUM  = 1.
      DO j = 1,NPROG
         SUM = 1. + j * RHO * SUM
      END DO
      RELTIME = NPROG / ((1. - 1./SUM) * CRITPROG)
      RETURN
      END
```

```
C=======================================================
C     CRITERION FUNCTION (AVERAGE RELATIVE ERROR)       |
C=======================================================
      FUNCTION CRITERION (CRITPROG)
      COMMON NPROG, TIME(20)
      C = 0.
      DO i = 1, NPROG
         C = C + ABS(TIME(i) - RELTIME(i,CRITPROG)) / TIME(i)
      END DO
      CRITERION = 100. * C / NPROG
      RETURN
      END

C=======================================================
C     COMPUTATION OF THE CRITICAL DEGREE OF MULTIPROGRAMMING |
C=======================================================
      EXTERNAL CRITERION
      COMMON NPROG, TIME(20)
      DO WHILE (.TRUE.)
         PRINT 10
         READA, NPROG, (TIME(i), i=1,NPROG)
         TMONO = TIME(1)
         DO i = 1, NPROG
            TIME(i) = TIME(i) / TMONO
         END DO
         CRITPROG = 10.
         CALL FMIN1 (CRITERION, 0.5, 0.00001, CRITPROG, CRITMIN)
         PRINT 11, CRITPROG, CRITMIN, 100.*(1. - 1./CRITPROG)
         PRINT 12
         DO i = 1, NPROG
            RT = RELTIME(i, CRITPROG)
            ER = 100. * (RT - TIME(i)) / TIME(i)
            EF = 0.
            IF (i .GT. 1) EF = 100 * (i-TIME(i)) / (i-1)
            PRINT 13, i, TMONO*TIME(i), TMONO*RT, ER, i/TIME(i), EF
         END DO
         PRINT 12
      END DO
   10 FORMAT('1Enter  NPROG, TIME(1), TIME(2), ... , TIME(NPROG) :'/)
   11 FORMAT(//' Critical degree of multiprogramming =', F6.3 /
     ,' Average relative error =', F6.2, ' %'/
     ,' Multiprogramming efficiency limit =',F6.2,' %'/// 62('-')//
     ,' Degree of  Measured  Computed  Relative  TMONO  Multiprog.'/
     ,' multipro-  elapsed   elapsed   error     -----  efficiency'/
     ,' gramming   time      time      [ % ]     TMULTI   [ % ]'/
   12 FORMAT(62('-')) .
   13 FORMAT(I5,T12,F8.2,T23,F8.2,T34,F6.2,T44,F5.2,T54,F6.2)
      END
```

Listing 4

```
Enter  NPROG, TIME(1), TIME(2), ... , TIME(NPROG) :

12  17.613 17.664 17.723 17.805 17.907 18.015 18.134 18.363 18.482 18.691 18.902 19.307


Critical degree of multiprogramming =15.916
Average relative error =  0.20 %
Multiprogramming efficiency limit = 93.72 %
```

Degree of multipro-gramming	Measured elapsed time	Computed elapsed time	Relative error [%]	TMONO ----- TMULTI	Multiprog. efficiency [%]
1	17.61	17.61	0.00	1.00	0.00
2	17.66	17.68	0.10	1.99	99.71
3	17.72	17.76	0.21	2.98	99.69
4	17.81	17.85	0.24	3.96	99.64
5	17.91	17.95	0.22	4.92	99.58
6	18.01	18.06	0.23	5.87	99.54
7	18.13	18.18	0.27	6.80	99.51
8	18.36	18.33	-0.19	7.67	99.39
9	18.48	18.49	0.06	8.58	99.38
10	18.69	18.68	-0.04	9.42	99.32
11	18.90	18.90	0.00	10.25	99.27
12	19.31	19.16	-0.78	10.95	99.13

Listing 5

The cental processor throughput is Up(n)/Sp and consequently the system throughput is now

$$X(n) = p1\ Up(n) / Sp = Up(n) / (1+Vd)Sp.$$

The number of jobs in the central subsystem n is a function of the number of terminals N. It can be obtained from the condition (R+Z)X(n)=N, i.e.

$$n = R\ X(N) = N - Z\ X(N) .$$

Finally, the average respose time of the system is

$$R = N/X(n) - Z .$$

The presented balanced central server model can be used for a detailed performance analysis and prediction of interactive systems at an intermediate level of detail where the modeling of disk subsystem is included. The technique is similar to the technique used for the machine repairman model. First, we use ITPB or IPDB for the measurement of interaction cycle time and processor utilization for various numbers of active terminals. Then the BCSM is used and its basic parameters Sp, Sd, and Z are adjusted so to optimally fit the measured results. After such a calibration procedure the BCSM is ready to be used for solving various performance analysis and prediction problems.

R E F E R E N C E S

BEN75 Benwell, N. (Ed.), "Benchmarking," John Wiley, New York, 1975.

BUZ71 Buzen, J. P., "Queueing Network Models of Multiprogramming," PhD Thesis, Harvard Univ., Cambridge, Mass. (NTIS #AD 731575, Aug. 1971).

BUZ73 Buzen, J. P., "Computational Algorithms for Closed Queueing Networks with Exponential Servers," CACM, Vol. 16, No. 9, 1973, pp. 527-531.

CUR84 Curram, J. and C.H.C. Leung, "Measurement and Modelling of an Interactive Distributed System Using Operational Analysis." Computer Performance, Vol 5. No. 3, September 1984, pp.178-181.

DEN78 Denning, P. J. and J. P. Buzen, "The Operational Analysis of Queueing Network Models," ACM Computing Surveys, Vol. 10, No. 3, September 1978, pp. 225-261.

FER83 Ferrari, D., G. Serazzi, and A. Zeigner, "Measurement and Tuning of Computer Systems." Prentice-Hall, 1983.

KLE68 Kleinock, L., "Certain Analytic Results for Time-Shared Processors," Proceedings of the IFIP Congress 1968, North-Holland, Amsterdam, pp. 838-845.

KLE75 Kleinrock, L., "Queueing Systems," (Vol. I and Vol. II). John Wiley, 1975 and 1976.

LAZ84 Lazowska, E.D, J. Zahorjan, G.S. Graham, and K.C. Sevcik, "Quantitative System Performance." Prentice-Hall, 1984.

SCH65 Scherr, A. L., "An Analysis of Time-Shared Computer Systems," MAC-TR-18, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1965.

SPO79 Spooner, C. R., "Benchmarking Interactive Systems: Producing the Software," Proceedings of the Conference on Simulation, Measurement and Modeling of Computer Systems, Boulder, Colorado, 1979, pp. 249-257.

ARTIFICIAL UNINTELLIGENCE

THE COMPUTER IN EDUCATION

John Higgins

Lancaster University

UDK: 519.682.7

ABSTRACT

Many applications of computers in education fail because they assume that learners, who may lack knowledge or skill, are unintelligent. In fact learners bring common sense and intelligent processing strategies to learning tasks, and do not always need an intelligent teacher, whether human or mechanical. A rule-bound program can provide a challenge to learners to discover rules, and the rules thus learned may be more memorable than rules presented as statements. This is illustrated with a language learning program called JACKASS.

Groupwork

I would like to start this session by describing a fairly well known computer program and saying what is likely to happen when a group of learners use it. Notice that I say a group of learners. If there is one thing that current experience with micro-computers should have taught us, it is the strength of the instinct to form groups and to collaborate on tasks. Much of our conventional educational practice is based on competition, the notion that the whole class are given the same exposure but then compete as individuals to show how well they have absorbed what they have heard. We praise and blame, pass and fail, create ranking lists on which somebody must be best and somebody worst. Introduce a computer and often this kind of thinking vanishes. The computer itself is now the only 'enemy', and then only because it cannot be influenced or wheedled into breaking natural rules. It is as heartless as nature itself. But, by working together to discover the rules it uses and then to turn them to advantage in solving problems, one can master the machine and master the rules.

JACKASS

The program I want to describe is one known in its original form as ANIMAL. I believe the credit for the idea belongs to Arthur Luehrmann and David Ahl, but it has had a number of re-incarnations under different titles and in slightly different forms. One of these is my own JACKASS, written for the Sharp MZ80A, the Newbrain, and the Spectrum.

The computer begins by challenging the learner to a guessing game: the learner is to think of something, and then the computer will try to guess what it is. In my version the starting categories are Animals, Objects, Cities or People. (There are other forms of the program, such as Acornsoft's TREE OF KNOWLEDGE, which allow the user to create new categories.)

Let us suppose that the user chooses Animals. A session of play might go like this.

```
DOES YOUR ANIMAL LIVE ON A FARM?

YES.

I KNOW.  IT'S A COW.  IS THAT RIGHT?

NO.

YOU KNOW MORE THAN I DO.  PLEASE TELL ME
THE NAME OF YOUR ANIMAL.

HORSE.

THANK YOU.  NOW PLEASE GIVE ME A QUESTION
BEGINNING 'DOES IT' TO DISTINGUISH A HORSE
FROM A COW.

DOES IT RUN RACES?

FOR A HORSE WHAT IS THE ANSWER?

YES.

THANK YOU.  NOW I KNOW 3 DIFFERENT
ANIMALS.  SHALL WE PLAY AGAIN?

YES.

THINK OF AN ANIMAL.  DOES YOUR ANIMAL LIVE
ON A FARM?

YES.

DOES IT RUN RACES?

NO.

I KNOW.  IT'S A COW.  IS THAT RIGHT?

NO

YOU KNOW MORE THAN I DO.  PLEASE TELL ME
THE NAME OF YOUR ANIMAL.

PIG.

and so on.
```

The machine is given one starting question, DOES YOUR ANIMAL LIVE ON A FARM?, and two animals, COW and ELEPHANT. This is the sum of its knowledge of the universe: anything which is not a cow is an elephant, and anything which is not an elephant is a cow. It falls to the user to teach it, by means of the guessing game, that the universe contains other animals. Every time the machine 'guesses' wrong, it adds the new animal and the new polar question to its database, and appears to become 'cleverer'.

Is it in fact getting cleverer? Obviously not. The algorithm it contains allows it to elaborate the nodes on its knowledge 'tree', but it is entirely dependent on the human to teach it the truth and indeed to provide it with consistent logic. There is nothing to stop the user from entering CAT in several places, ie as an animal which lives on a farm and again as an animal which does not live on a farm. The machine will not, given the present form of program, recognise the inconsistency. There is nothing forcing the human to enter real animals: fantasies and nonsense are just as readily processed. Experience suggests, however, that users strive towards logic, even when they are dealing in fantasy: unicorns, griffons, and bats-out-of-hell may be entered, but they are described and classified as logically as possible.

This may partly be due to what happens at the next stage of the program, once a number of new entries have been made. As soon as the user answers NO to the question SHALL WE PLAY AGAIN?, the display changes to the following menu of options:

1 TEST ME ON WHAT I KNOW

2 LET ME TEST YOU

3 SEE A CLASSIFIED LIST

4 GO BACK TO THE GUESSING GAME

5 SAVE THE GAME ON TAPE

6 END

The first option represents the user's reward for having taught the machine, since it requires the machine to use its classified knowledge to present little 'essays' about each animal. First the user selects an animal from a displayed list by means of a pointer. Now the machine will display something like the following:

THE ELEPHANT

..... LET ME THINK.

AN ELEPHANT IS AN ANIMAL. IT DOESN'T HAVE STRIPES, DOESN'T EAT NUTS, HAS A TRUNK AND DOESN'T LIVE ON A FARM.

If the second option is selected, the display will be

THE ANIMAL I AM THINKING OF DOESN'T HAVE STRIPES, DOESN'T EAT NUTS, HAS A TRUNK AND DOESN'T LIVE ON A FARM. WHAT IS IT?

and now the user can point at an animal and get the answer THAT'S RIGHT or I WAS THINKING OF AN ELEPHANT as the case may be.

Morphology

The computer's composition skills are rather limited in this form of the program. It simply goes from the animal node back up the question tree, selecting either DOESN'T plus the verb stem plus the rest of the phrase if it is climbing a negative branch, or verb stem plus -S plus the rest of the phrase if it is climbing an affirmative branch. However, even that process is not as simple as it sounds, since adding a third person -S to an English verb is subject to some interesting morphological constraints. The first challenge one can give to the learners is to find out if the machine knows all the constraints. Can the class catch the computer out, force it to make a mistake? In an early distributed version of the program there was a programming error (a missing RETURN) which meant that verbs like FLY were turned into FLYSES. This was amusing, but quickly led to the question, what would the program do to SAY or PLAY. It got those right. So what kind of error was involved? And what was the rule of English grammar which the program should have used?

The program made use of one other set of morphological rules in order to put AN in front of ELEPHANT and A in front of COW. The rules can be expressed fairly compactly, but not many native speakers of English could state them all without a lot of thought. Consider the cases of UNICORN, UNEXPLORED TERRITORY, HOUR, ONE-TOED SLOTH, and ONAGER. The subroutines which I used to select A or AN and to add -S to verbs derived from what were originally free-standing programs for the 1K ZX81 computer, written by my colleague Tim Johns. He used them purely as devices to allow students to explore the

morphological rules involved. I deliberately left a few 'holes' in my versions, so that students might try to catch the machine out. In the case of A/AN, for instance, I omitted to 'trap' the case of words or phrases beginning with ONE or ONCE. Although FLYSES was just a programming error at first, I have on occasion used the faulty version deliberately in order to get students to think about the possible causes.

Adverbs of frequency

Another, more important, omission in the -S ending routine is that I have not tried to trap any frequency adverbs. If the user's question is DOES IT EVER EAT BAMBOO?, the machine's statement will be either IT EVERS EAT BAMBOO or IT DOESN'T EVER EAT BAMBOO. It is interesting to note that the negative sentence is acceptable. If the input question had been DOES IT SOMETIMES CATCH MICE?, then both versions of the statement would have been faulty: IT SOMETIMESES CATCH MICE and IT DOESN'T SOMETIMES CATCH MICE.

This particular bug could be caught easily enough by providing a list of frequency adverbs which the machine should match against the first word after DOES if, but having made the match, it is still interesting to consider what the machine should do when it turns the questions into statements. Consider these examples:

```
DOES IT EVER CATCH MICE?
YES:  IT SOMETIMES CATCHES MICE.
NO:   IT NEVER CATCHES MICE.

DOES IT OFTEN CATCH MICE?
YES:  IT OFTEN CATCHES MICE.
NO:   IT NEVER CATCHES MICE.

DOES IT SOMETIMES CATCH MICE?
YES:  IT SOMETIMES CATCHES MICE.
NO:   IT NEVER CATCHES MICE.

DOES IT USUALLY CATCH MICE?
YES:  IT USUALLY CATCHES MICE.
NO:   IT DOESN'T USUALLY CATCH MICE.
```

There is much interesting material for students to speculate on here. What, for instance, is the opposite of ALWAYS? Is it NEVER or SOMETIMES? The answer is not to be found in language rules but in real world truths or in the speaker's mind.

The next stage of improving the program would be to equip it to break up its little essays into several sentences. We could, for instance, instruct it to put two 'facts' into each sentence, linking them with AND if they are both affirmative or both negative, or with BUT if there is one affirmative and one negative statement. The description of the elephant would now come out as follows:

```
THE ELEPHANT

AN ELEPHANT IS AN ANIMAL.  IT DOESN'T HAVE
STRIPES AND DOESN'T EAT NUTS.  IT HAS A
TRUNK BUT DOESN'T LIVE ON A FARM.
```

One could equip the program with a reduction rule which will remove repeated verbs in the same sentence, thus turning

```
IT HAS STRIPES AND HAS CLAWS.
```

into

```
IT HAS STRIPES AND CLAWS.
```

If the machine is given this rule to apply in all cases, however, it will turn

```
IT DOESN'T HAVE STRIPES AND DOESN'T HAVE
CLAWS.
```

into

```
IT DOESN'T HAVE STRIPES AND CLAWS.
```

suggesting that STRIPES and CLAWS are a closely linked pair. Sooner or later, a student should point out that OR is safer to use when a negative has been deleted. Another interesting point that should surface in discussion is that the affirmative-negative pattern can be shortened as follows:

```
IT HAS STRIPES BUT NOT CLAWS.
```

whereas the negative-affirmative pattern will not work:

```
IT DOESN'T HAVE STRIPES BUT CLAWS.
```

This is a possible sentence, but suggests that claws function as a substitute for stripes, and that no animal could have both.

This could lead learners to wonder if there is something a bit odd about the last sentence. In general we allow disconnected senses to be concatenated with AND but we expect a stronger thematic link when two notions are linked by BUT. The sentence

```
MY SISTER CAN'T PLAY THE PIANO BUT MY
BROTHER LIKES CHEESE.
```

(which is said to have been printed in a language teaching textbook) rightly aroused mirth. The program, if it performs algorithmically, is bound to generate a large

number of such pairs. What can then happen is that learners start to enjoy themselves looking for ludicrous pairings:

IT DOESN'T LIVE IN THE SEA BUT IT EATS RABBITS.

IT LIKES PEOPLE BUT IT DOESN'T FLY.

What are the implications of these? The first suggests that only sea-creatures normally eat rabbits, and the second that only flying creatures like people.

With some effort it would be possible to anticipate many of these problems and equip the program with enough artificial intelligence (eg through a semantically tagged lexicon of verbs) to filter out all the nonsense. But what would be the gain? It would mean that the learner would never be able to teach the machine falsehoods or nonsense, and would never see falsehoods or nonsense on the screen. At first sight this might seem to be something to applaud. On reflection, however, surely this implies very little respect for the learners' own common sense, and imposes vast restrictions on the learners' imagination. The mere fact that the machine carries out orders in a slave-like and completely unimaginative way can be a liberating factor when a human being comes to use it. There are times when the machine's lack of intelligence shows us things we might never have noticed for ourselves and awakens intelligence and imagination in people who have had little chance to develop them before.

Human intelligence

It is for this reason that I am unhappy when people talk about intelligent tutoring systems. I agree that much of the material that passes as educational nowadays is deplorable and unimaginative. I am talking about tutorial dialogue, in which the computer makes statements and sets questions. Those answers are then judged right or wrong, and scores are calculated. Here the computer conceals its unintelligence behind a set of messages which seem to claim authority and intelligence (in fact those of the program author rather than the machine). The results achieved by such programs tend to be disappointing, and it might be claimed that we need to equip the machines with both **knowledge** (in order to deal with unanticipated questions) and with a **mental model** of the student in order to diagnose the reasons for wrong answers and so give appropriate explanations. These 'intelligent tutoring systems' will demand vast memories and therefore elaborate machines. But since the learners themselves are already bringing with them to the learning situation a great deal of intelligence and knowledge, I see no reason to wait for such systems. Game-like programs running on small micros can provide rule-governed environments in which certain kinds of learning can take place very easily. By respecting the learners' intelligence and providing this kind of experience, we can perhaps make many kinds of learning more rewarding and more fun.

ESPRIT Project 510 "TOOLUSE"
DESCRIPTION, PROGRESS TO DATE & FUTURE DIRECTIONS: A VIEW FROM THE TRENCHES

James A. Redmond

(Dept. Of Computer Science, Trinity College, Dublin 2, Ireland)

UDK: 681.3.06

Abstract
Project ToolUse aims at giving active assistance in the design, development, implementation, evolution and reuse of packaged software. A major feature is the focus on software development methods as objects to be manipulated. This will be facilitated by the development of a prototype environment supporting a development language and an underlying knowledge base. ToolUse will use formal methods wherever possible, but if necessary will use semi-formal methods to achieve the project goals and objectives.

1.0 INTRODUCTION

ESPRIT Project 510 formally entitled "An Advanced Support Environment for Method-driven Development and Evolution of Packaged Software" is 50% funded by the Commission of the European Economic Communities, and has teams from industry and universities in Belgium, France, Germany and Ireland. The project is more informally known as the "ToolUse" project. The project budget is 12.45m ECU (about $10m.) and it will last 5 years. The official start date was October 15, 1984. The teams involved are:

BIOMATIK - Freiburg, West Germany
CERT - Centre d'Etudes et de Recherches,Toulouse,France
GENERICS - Generics (Software) Ltd, Dublin, Ireland
GMD - GMD Forschungstelle, Karlsruhe, West Germany
I.I. - Informatique Internationale, Toulouse, France
INSIGHT - Insight Software, Dublin, Ireland
TCD - Trinity College, Dublin, Ireland
UCL - Universite Catholique, Louvain, Belgium

This project aims at providing active assistance in the design, implementation and evolution of software. Such assistance should be obtained by describing the development of software in a formalized manner such that this description can be manipulated. The development of such a formalised description can only be based on a thorough understanding and formal definition of methods driving the process of software construction from the first description within a framework of an application-oriented specification language to the stages of implementation, use and continued evolution and reuse. The use of a formalized description must rely on the existence of an advanced support environment. This support environment in turn must comprise tools relying on the understanding, acquisition, representation and reuse of knowledge and constraints related to information processing techniques depending on given application areas and on the target system.

The project will provide a prototype environment containing an adequate user interface supporting the use of the development language, an underlying database for the administration of data representations and a set of tools supporting the whole development process. The short-term goal is to prove the feasibility and provide the requirements specification for the environment. Progress to date is discussed.

2.0 UNDERLYING FOUNDATIONS

2.1 The Traditional Approach

Software development can be regarded in simple terms as in Figure 1:

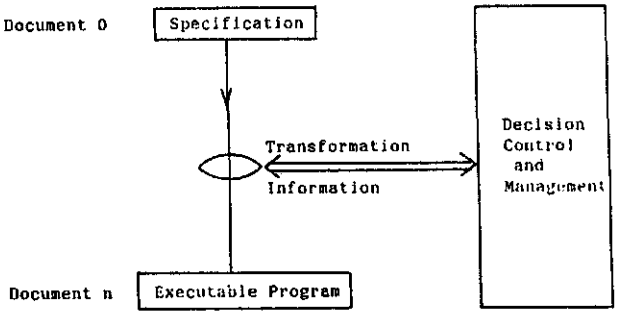


Figure 1 Software Development in simple terms

The most serious problem with this view of the software development process is that a general-purpose automatic programming transformer which will take a specification and generate an executable program from it does not exist, and is unlikely to do so in the future.
So therefore it is necessary to break the process up into a number of steps. The traditional approach is the classic waterfall diagram of Boehm [Boehm76]. This is shown in Figure 2, which is modified after a diagram by G Koch at a ToolUse meeting in October 84.

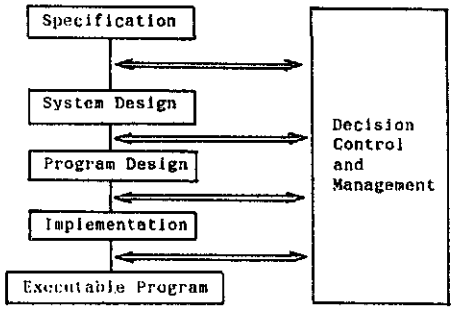


Figure 2 Traditional Approach to Software Development

Note that some steps in this process have been automated (e.g. compilation), while most steps are manual, with some automatic support. The major problem with this approach is the QUALITY of the final product - errors still persist in the final product (e.g. a rule of thumb in industry is that five bugs per thousand lines of code exist after implementation). Quality of product can be measured in different ways, e.g. reliability, usability, functionality, maintainability, efficiency, flexibility, user-friendliness etc. However CORRECTNESS dominates all others. In general, it is possible to suffer an inefficient, user-unfriendly system which is correct, but not one that gives inaccurate answers.
To improve this aspect of quality, the traditional direct solution has been to use TESTING. The major criticism of testing is that testing is unable to prove the absence of bugs. However the main affect of testing is to increase the CREDIBILITY of a particular program. An indirect method to improve quality has been to enforce particular design and programming METHODOLOGIES. These approaches are shown in Figure 3. The affect of these approaches is to increase the credibility of the final product.

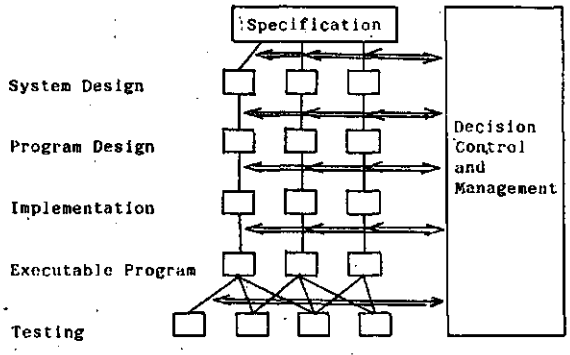


Figure 3 Ways of Increasing Credibility

In Figure 3, it may or may not be decided to use a particular methodology and/or testing. There are different pathways through the graph. Using a good methodology AND testing would increase the credibility of a program product. This product will certainly still have bugs in it. These will show up in operation, and will be handled/ corrected/ reprogrammed under what is quaintly, but inaccurately called "MAINTENANCE". In fact, while a small percentage of maintenance time (typically 20% or less) is spent in correcting bugs, in general it involves MODIFYING the system in response to changes in the user's environment. Maintenance is an extremely costly activity for a number of reasons, not least to do with the fact that the person who is maintaining the system say 5 years later is nearly always NOT the original designer. To ease his job and to forestall the introduction of even more errors, it would be useful to give him access to a DESIGN HISTORY of the original program/system. The most important information this history would give is why certain design decisions were taken, and why other decisions were rejected.
In certain circumstances, the less credible alternative in Figure 3 may be preferred. For instance in the case of RAPID PROTYPING where a product is developed quickly with little regard for error-handling, user-friendliness etc., but where the essential functionality is presented so as to get feedback from the user as to whether the original specification is adequate or not.
So far it has been assumed that the specification captures the requirements of the user community. In fact, this is highly unlikely, and it is much more likely that the user does not know exactly what he wants until a system is delivered to him which he can see, touch and experiment with. This "TANGIBILITY" almost immediately tells him what he does NOT want, but this is usually what has just been delivered.

It appears therefore that Figure 3 is over-simplified in terms of acquiring a correct specification of the user's requirements, and should be extended somewhat as follows in Figure 4:

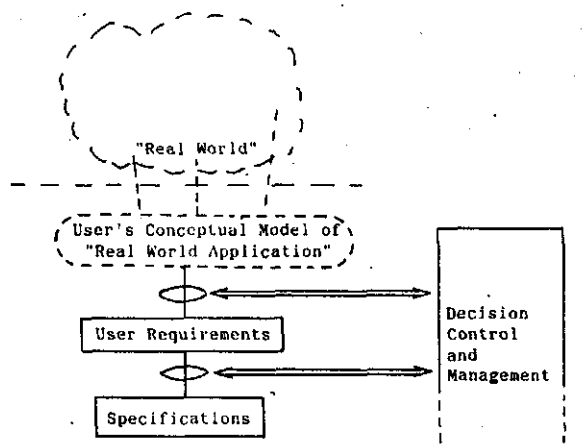


Figure 4: Acquisition of Specifications from a UCM.

In this diagram the acquisition of requirements from a user's conceptual model of the real world application may be manual, semi-automatic or automatic. In general these requirements are informal, imprecise and incomplete but may be made more formal, more precise and more complete with the aid of tools [BORG85]. In fact, the requirements document itself is a simplification in that feedback (for instance using the "tangibility" of rapid prototyping) is necessary to make the requirements more concrete (vid Figure 5).

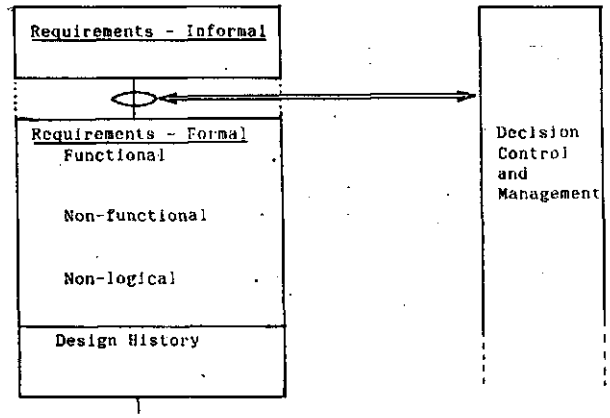


Figure 5 Making the Specification More Complete

For each transformation in Figures 4 and 5, it should be possible to do as much as possible automatically, but if this is not possible the Developer should be allowed as much information as possible to allow an intelligent choice of transformation, e.g. in Figure 3 whether to use an optimizing compiler or not. In Figure 3 the possibility of going down the prototyping route after some stages of the structured approach is not precluded, e.g. for a prototype one might not want to do any testing.

Industry has used these now traditional techniques for the last ten years or more i.e. they are happy with a structured methodology and a standard approach to testing, possibly with the customer signing off on the last acceptance test. This works for large scale projects; they are routinely implementing systems which would have made headlines ten years ago. They feel that they can live with the present technology; typically they will have evaluated formal methods in the past and are not impressed particularly from an economic

viewpoint. Barry Boehm has been heard to say that he has yet to see a formal system implementation that did not take at least five times as long as a conventional approach would have taken. The ToolUse project emphasises formal methods whenever possible, semi-formal methods will be used otherwise. This discussion is an indication of the credibility gap that formal plus semi-formal methods have to bridge. A further indirect method of increasing the credibility of the process is automating as many manual tasks as possible, and providing support for the creativity of the programmer in a programming support environment. Another inadequacy of the exposition so far is that the focus has been on a tested, executable product but the characteristics of the TARGET SYSTEM (i.e. the software-hardware system) on which it is to run has been ignored.

The next section discusses one formal approach which has been developing for at least the last fifteen years, and has recently been looking much more promising.

2.2 The Transformational Approach

What has been described so far is essentially one conception (industry's) of "Programming-in-the Large" and what corresponds to industrial practice today. Another much more theoretically acceptable approach to the software development problem shown in Figure 1, is that used by Program Transformation Systems [BALZ81] [WILE83] [PART83]. This approach reflects the idea that the difficulties involved in constructing correct programs can only be overcome if the whole task is broken down into sufficiently small and formally justified steps. The essential idea is to modify Figure 1 as follows:

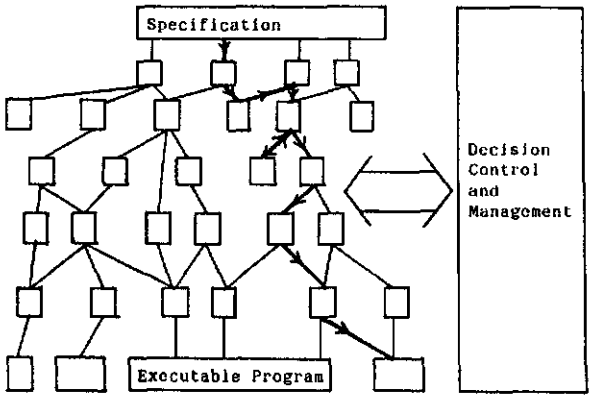


Figure 6: The Transformational Approach

A possible solution has been darkened in. Notice that backtracking may occur. All transformations ideally could be done automatically. In practice human decision control and management of the process is necessary.

Transformational programming is a methodology of program construction by successive applications of transformational rules. The individual transitions between the various versions of a program are made by applying correctness-preserving transformation rules. It is guaranteed that the final version of the program will satisfy the initial specification. This approach has the enormous advantage of solving the credibility problem, but the problem of efficiency is a very large problem with this method. Nevertheless some transformational systems can even take a tentative, incomplete specification and gradually develop it into a complete one.

Nearly all transformation systems are interactive and rely interactively on the developer to resolve unexpected events. Most systems document the development process in some fashion ranging from a simple, sequential log to sophisticated database mechanisms. The most clever are even able to automatically redo a particular program development starting from a (slightly) changed input. Most systems have a predefined collection of transformation rules, either as a catalogue (with problems of completeness and structures) or a generative set (a small set of powerful elementary rules to be used as a basis for constructing new rules).

The major problems with transformational systems have been their efficiency, their narrow scope and their failure to scale up. Programming in the Large in academic computer science is building up large programs from simpler small ones. This has not been successful (yet?) and there has been no significant acceptance yet into industry, although this has been forecast for some time.
"The time is clearly ripe for
program manipulation systems"
This comment by Knuth in 1976 would appear to have been somewhat premature. A variety of systems exist in research institutes today [PART83]. However none yet work on other than "toy" problems of narrow range. However, work is under way worldwide to investigate the unsolved theoretical problems and also to check the methodology's feasibility on medium-size problems from many application areas.

2.3 The Inferential Programming Approach

Both previous techniques have their advantages and disadvantages - large, efficient programs, but not proven correct, versus small, correct programs, in a narrow application range, but inefficient. Inferential programming, referred to by Scherlis and Scott [Scher83], would seem to offer a reasonable compromise, but with formidable obstacles to be surmounted.

They feel that what is needed is a logic of programming. This is a mathematical system which allows reasoning about individual programs, but also the carrying out of operations (transformations) on programs, as well as the reasoning about these operations. Their fundamental concepts are:
program derivation: the conceptual history of a program
inferential programming: the collection of methods and associated tools for constructing, modifying and reasoning about such derivations over time.

They feel that it is much more useful to talk about a program in terms of its EVOLUTION, i.e. the sequence of insights which is required to derive the implementation from a specification. By representing the process of program development as a sequence of programs, arranged as if the final implementation were developed from an initial specification by a series of refinement steps, we can maintain a structure in which clarity, modifiability and efficiency co-exist. Therefore the programmer does not have to worry about for instance an ADT and its efficient implementation in the same program. A specification and an implementation differ only in degree, and may be regarded as being on the same axis of structural choice. One end of this axis represents a specification which is for human consumption, while the other end represents an implementation which is for machine consumption.

Inferential Programming is analogous to the process of building mathematical proofs. Mathematicians do NOT develop proofs by starting at line 1 and slogging through to the last line. Instead, they formulate a general strategy for the proof and try to fill in gaps in it until they have enough detail to make a convincing argument. The proof that subsequently emerges is a HIGHLY-STRUCTURED justification for a mathematical fact, even if it is written in ordinary language. On the other hand, the process of building a proof is a somewhat undisciplined and exploratory activity, in which insights are gained and expressed, and finally woven together to form a mathematical argument. Similarly a program derivation is a highly-structured justification for a program, while inferential programming is the process of building, manipulating and reasoning about program derivations, and is analogous to the proof building process.

Inferential programming however will allow proofs and programs to be developed together without requiring the programmer to rediscover the insights that went into the original development of the program as in conventional correctness proofs. By representing programs as sequences of derivation steps and using systematic techniques to move from one step to the next, the correctness of the program will follow directly from the correctness of the derivation techniques. With cleverly constructed mechanical tools, the business of proof could be so effectively hidden from users that the development of a correctness proof would appear to be automatic. Also those steps in a derivation whose correctness has not yet been proven can be isolated to facilitate debugging and testing.

The conventional wisdom is that it is better to modify an old program, than to develop an entirely new program [SILV85], therefore significant sharing of the program derivations is to be expected, e.g. compiler examples in [Fren85]. Modification is difficult in a conventional framework, because, like a posteriori verification, it requires rediscovery of concepts used during development of an implementation. Simple conceptual changes to a program often require complicated and extensive changes to code. In an inferential programming system, the conceptual changes can be made at the appropriate places in the program derivation, and the supporting system can be used to propagate these changes correctly into implementation.

The success of programming logics with large problems has been minimal. The question must also be asked of inferential programming - will it FAIL to scale up? Modularization should help, but the question remains valid and open.

3.0 THE REQUIREMENTS FOR A SOFTWARE DEVELOPMENT ENVIRONMENT

A discussion has been given of the traditional software engineering approach as practiced by industry, the formal approach favoured by academia and research institutes, and inferential programming which is an attempt to bridge the gap between the two. Inferential programming is closer to the underlying philosophy of ToolUse. Based on this discussion, it is possible to specify the ideal software development system. A software development environment with a solid formal basis is required. This is an environment in which programs are constructed by formal development methods and which supports formal and well-understood software development principles. By formal methods is meant those which have a sound theoretical basis where software specifications, designs, developments and implementations are treated as mathematical objects whose properties can be investigated by formal analysis. In contrast, in informal methods the semantic properties of objects are usually described in informal natural language terms, which makes it difficult to manipulate them as objects.

A software development environment is required for large scale problems. The following are its desirable characteristics:

- The basic OBJECTS to be manipulated, and their required behaviours, should be represented by a mathematically based theory or model. This theory should be powerful enough for real world systems and ideally should cope with sequential, non-deterministic and concurrent aspects of systems.
- There should be a CALCULUS for manipulating formal objects which represent system components and their behaviours. Usually the operators are composition and decomposition and possibly higher level operators which may assist in the process of refinement by ensuring correctness-preservation during a refinement step.
- The software engineer should be supported with automatic SUPPORT TOOLS as much as possible, e.g. syntax checking, type checking, documentation, theorem proving, correctness preserving assistance during transformation, prototyping etc. There should also be support for partitioning large problems, e.g. version management etc.
- Support should be provided for EACH STAGE of the life cycle.
- The formal methodology should support VERIFICATION and VALIDATION, where verification is a formal rigorous proof that a design satisfies a specification while validation relies on a convincing demonstration that a formal specification captures a system requirement or that a specification is acceptable to the user.
- A formal methodology should handle REQUIREMENTS ANALYSIS as an iterative process during which an analyst attempts to construct a formal model encapsulating the user's requirements. The problem areas are consistency, completeness and feasibility. Ideally ways of animating the model so as to display the properties of the specification to the user should be available.
- Aid for software reuse should be provided
- Significant support (training, packages, case studies etc.) should be available for the methodology to allow it to be marketed to industry.
- The formal methodology should also SCALE UP. This has not happened in the past with large projects. Where it is not possible to use a thoroughly formal framework a "rigorous" approach should be used. This approach may appeal to intuition in the presentation of correctness arguments. The formal framework allows formal verification to be pursued in the event of a query.

However in practice the situation is even more depressing. It is not yet practical to adopt even a rigorous approach. Therefore the alternative is to mix formal and semi-formal approaches. This mixture of formal and semi-formal approaches could be a mixture of Figures 3 and 6, for instance getting from requirements to a specification might require a semi-formal method, while from a specification to an implementation could be handled by a transformational approach. While not ideal this has obvious significant advantages and may be essential for convincing a sceptical industry. The use of formal and semi-formal techniques in electrical and electronic engineering is a historical example of this approach. There is the possibility of initially starting in a semi-formal fashion and increasing the formality as necessary with time.

Unfortunately for a good software engineering method, more than just a sound theoretical basis is needed. Industry remains very sceptical that tangible practical benefits will accrue from implementing these methods. There is an abundance of elegant mathematical theory. Turski [TURS84] has commented that there is a danger that the mathematics may be pursued as an end in itself (since it is elegant and relatively easy) while the engineering problems are ignored. Guttag et al. [GUTT82] have stood back and criticized their own research and concluded that it is:
- difficult to establish a useful correspondence between elegant formal systems and the untidy realities of writing and maintaining programs.
- difficult to maintain a consistent attitude about what constitutes a "good" specification.
- too easy to become bogged down in the clerical details of managing, reasoning about, and maintaining the consistency of large specifications.

The traditional semi-formal design methods are functional decomposition, stepwise refinement, data decomposition. The most successful formal methods to date have had an approach similar to these (e.g. JSD, VDM). Semi-formal methods generally support the construction of system models. Model-oriented approaches are more easily assimilated by software engineers. So in a model-oriented approach, a rigorous approach is attractive since it allows the use of justifications which appeal to experience or intuition within a formal framework. If a formal method is used along with the traditional semi-formal techniques, then formality can be introduced gradually.

4.0 PRINCIPLES AND GENERAL DESCRIPTION OF TOOLUSE

The main motivations of ToolUse are:
- the understanding and expression of software design methods

- the modelling and use of application domains and target systems knowledge.
- the design and experimentation with a prototype software environment.

The general principles underlying ToolUse are:
- the software development process is described by distinct states, each associated with information about the software product.
- state transitions can be described/ aided via a development language.
- traversing the states from specification to implementation, via the use of correctness-preserving transformations, guarantees the logical consistency of the products.
- at every state, knowledge can be acquired or acquired knowledge can be reused.

The main research and experimentation areas are:
- the understanding and formal modelling of:
  - the software development process
  - application domains
  - target systems
- the specification, definition and evolution of a prototype development language.
- the definition, implementation and evolution of a prototype environment supporting the development language.

4.1 Method-Driven Software Development

The objective is to gain a deeper understanding of the process of software development, and what means are suitable for describing the process of software development. Software development and evolution consists of writing specifications and transforming them into programs by the following steps:
- representation of data types by other data types
- introduction of control structures
- integration of predefined parts
- specialisation or generalisation, via instantiation or parameterisation.
- transitions between distinct representations

Software development and evolution should be organised in terms of compositions using the composition rules: combination of independent steps; choice between alternative steps, and the abstraction of design schemes into functions.

The Logical Structure of Development Methods
Many informal methods are currently used to express the software development process. ToolUse will investigate the possibility of reasoning about design decisions within the development language. The intention is to evaluate to what extent and by what means it is possible to use design methods as the main driving force in the construction of software support environments. Some criteria to be used in picking suitable development methods are:
- suitability for system composition (for large scale problems)
- adequacy for desired application domains and target systems
- continuity of designs with respect to evolution
- logical coherence between successive states
- possibility of using methods as parameters in the process

Knowledge Representation
ToolUse will make use of explicit knowledge representation. The methodologies for expressing knowledge representations will be analysed with respect to:
- meta-knowledge and development paradigms
- design strategies and transformation plans
- specialised expertise available to the project
- human comprehensibility
- adequacy for automatic processing

The Development Language
The following problems must be addressed:
- the overall structure of the development language
- the techniques to ensure coherence of the development steps (does the language need the intent of the design decisions?; and how will it control their consistent evolution?).
- the expressing of specifications, design methods and strategies in a suitable form for organising in libraries.

Basic Support for Program Development
There are possibly three main classes of tools:
- tools for assisting the environment configuration, according to specific development methods, application domains and target systems.
- tools supporting development methods and scenarios, e.g. "intelligent compilation" for instance a 4GL application being transformed to an imperative implementation, direct interpretation for rapid prototyping for instance.
- tools supporting the integration of components resulting from the development plans, e.g. administration of state information about program components; and assembly of systems out of their components.

Application Domains
The objective is to provide an experimental basis for the experiments with the final prototype environment. This basis is formed by a set of representative application examples covering real world problems. They are selected to demonstrate the applicability of methods, tools and languages supported by the prototype environment and the development language.

Modelling of Knowledge from Application Areas
The objectives are:
- classify applications
- develop means for describing applications
- describe and structure expert knowledge about the applications

Knowledge Transformation and Man-machine Communication
This is concerned with the communication between the application expert and the computer expert. The process of capturing requirements and mapping them into specifications needs methods for knowledge acquisition, structuring of specifications and building and evaluating prototypes as examples and experimental objects.

Target Systems - Reuse of Existing Components
Research on the topic "Target Systems" aims mainly at facilitating the reuse of existing components. This requires the availability of different formalisms for functional and operational specifications at the different levels of abstraction within the program development process. It is necessary to build models of the target environment.

4.2 Summary Of Key Concepts Of ToolUse

The previous discussion has served to highlight the major underlying concepts and motivations of ToolUse:
- Formal Methods where possible
- Knowledge-based Methods where necessary
- Method-driven development
- Concern with Application Domains
- Concern with Target Systems
- Development Methods and Plans as Objects
- Environmental Support
- Composition of Design Knowledge
- Vertical Integration i.e. consistent representations on different levels
- Integration of Tools
- Large-scale applications
- The higher level expression of design knowledge should be a guiding principle
- Modifiability and Reuse
- Adequate User Interface
- Toolset
- Developer's Assistant
- Development Language
- Nonfunctional requirements

5.0 DESCRIPTION OF TASKS AND PROGRESS SO FAR

The activities of the ToolUse project are organized into seven main tasks:
Task 1: Basic Support Tools
Task 2: Methods Evaluation
Task 3: Development Language
Task 4: Support Environment Synthesis
Task 5: Application case studies
Task 6: Target systems
Task 7: Logistic support and management

5.1 Task 1 (Basic Support Tools)

Task 1 has the objective of providing the basic support tools the prototype environment. To this end Task 1.1 evaluates tools from existing environments particularly with respect to how method-driven they are. T1.2 selects a basic toolset for use within the project.

Task 1.1 consists of 2 phases. The initial objective of Phase 1 was to identify the role of tools in existing environments especially:
- how method-driven or method independent a tool is.
- how applicable the tool is over a set of applications or target areas.

A decision was made at project start-up to associate Task 1.1 and Task 4.1 (Evaluation of existing environments) because of their obvious close affinities i.e. it was difficult to look at tools without also looking at environments. Some problems have been encountered in executing the original plan for Task 1.1.

PROBLEM 1: The realisation that tools were rarely separable from their environments - even in UNIX.
SOLUTION: It was decided to closely couple T1.1 and T4.1 so that:
T1.1 would study the function of tools in SEE's.
T4.1 would study the SEE's support of DESIGN and use of DESIGN KNOWLEDGE.

PROBLEM 2: How can the role of tools be assessed when they are so closely linked to environments?
SOLUTION: Environments were broadly classified into 3 classes, and the tools were characterised in each class. The 3 classes were:
- Programming Environments, e.g. UNIX, PCTE [BOUR85].
- Software Environments, e.g. Affirm [GERH80], SPRAC [LEMA83], Mentor [DONZ80], GRASPIN [HEYD85].
- Support Environments, e.g. (possibly CHI [GREE82]).

In general the less advanced environments have DISCRETE GENERAL PURPOSE tools whereas the more advanced (Support) environments are HIGHLY INTEGRATED so that the tool is CONCEPTUAL rather than a separate component.

Task 1 Results to Date
- Tools and Environments are NOT, in general, easily separated.
- Method support and reuse of design knowledge seem to require a tightly integrated environment.
- The study of tools as simple functional units will NOT be sufficient for ToolUse. Instead the role of tools must be identified within their home environment.
- The prototype environment will often use tool CONCEPTS rather than simple copies of useful tools.
- A Unix-like environment will be used for ToolUse project management.
- Work to date has concentrated on standardising tools and their usage at each partner's installation.
- Selection of tools will not be as simple as originally anticipated. Instead experience will be gained through using and porting a number of different environments.

Task 1 Future Work
The Tool/ Environment classification will in future focus on two main aspects:

FUNCTIONALITY: How does it contribute to:
- An Object Management System
  (The richness of the object increases from Unix to PCTE to Adele [ESPR86] to CHI).
- A Developer's Assistant, using
  - communications tools,
  - syntactic tools,
  - semantic tools
  - support tools

QUALITY:
- where used in SLC
- for which applications/targets
- which methods are supported
- how reusable is the tool or concept
- availability and documentation of tool

Short Term Workplan (6 Months)
The next 6 months involves close interaction with T4 and T5. Things to be done are:

1. Choose one or two substantial case studies.
2. Carry out a design on these using a small representative set of environments (Mentor, Promod, Sprac, Affirm, Interlisp).
3. Observe and record the use made of tools during this design work. In particular:
- how are data/ knowledge represented.
- how much does the tool SUPPORT transformations or ASSIST IN THE CHOICE of transformations.
- how reusable is the tool or tool concept.
- does the tool reflect a particular method, applications area, or target system.
4. Complete the new functionality/ quality classification for each tool.
5. Extract key tools/ functionalities and synthesise them to form part of the requirements for the prototype environment.
6. Produce the final report for Task 1.1

These studies will lead to a deeper understanding of the concept of tools, especially in the area of software knowledge management, and will enable a first set of requirements to be produced for the prototype support environment.

The major criticism of the work so far is that the merging of the tools/ environment study focussed too much attention on the environments to the detriment of tools. This is perfectly understandable as tools can only be understood within their environments, so it is natural to try to comprehend the environment first. Most classifications were done for environments, very few were for tools. A shotgun approach to the problem has been used where a rifle approach might have been more appropriate. To clarify this, we are much more interested in nob-obvious tools or tool concepts which might be useful in the rest of the project.

Longer Term - Task 1.1 -- Phase 2
Task 1.1 does not end after the first year but comes alive formally again after month 30 and lasts until the end of the project. The objectives of this phase are:
- Review new tools and concepts as they appear
- Incorporate these developments into the prototype environment.
- Continually reassess and develop the ToolUse toolset.

5.2 Task 2: Methods Evaluation

The overall goal of Task 2 is to compare various important software design methods in order to find a set of concepts expressing the logic of these design methods. Research should also focus on finding a suitable notation useful for formulating these concepts. This notation could be a first stab at a subset of DEVA, or at least at what DEVA should express formally. DEVA is the name for the development language of the prototype environment to be built by the project. The goal of Task 2 is to compare various design methods and amalgamate them where and if possible; then extract any general underlying methods as a guide to what the primitive concepts in DEVA should be.

Initially two well-known and quite different methods, the Vienna Design Method (VDM) [JONE79] and Jackson's method (Jackson Structured Design - JSD [JACK83]) were selected. (Jackson Structured Programming - JSP, a precursor and a subset of JSD was also investigated).

Jackson's Methods cover the whole development process, from informal requirements to programs written in a target language. They aim at providing significant support for developers, especially in building (formal) specifications from requirements. Both methods use the idea of data stream as a fundamental concept. JSD initially builds a model of the real world; this model is an "executable" specification of the problem. Then this specification is transformed for execution on a target system. The JSD paradigm is essentially "Model First". Jackson claims to be a method based on composition, not decomposition. It is important to note that JSD and JSP have been widely used commercially for quite some time.

The strongest contribution of VDM to the development activity seems to be that it forces the designer to use a well-established formal (but not executable) specification language, based on a formal mathematical underpinning. The success of VDM is probably linked to the importance of a good, flexible notation to support mental activity together with "rigorous" checking of development steps. The algorithmic part of the method (i.e. how, why and when to connect different development steps) is much less formal and seems to be actually used much less. The general idea is a topdown hierarchical approach to the development, in which subproblem specifications are progressively produced. VDM has been used generally in research projects, but has not in any way received industrial acceptance.

In future, more feedback will be solicited from partners who have direct experience with the methods of interest on real case studies, in order to deepen the technical understanding of the basic mechanisms and to isolate pragmatic rules associated with given application areas. It is proposed in future to also look at SADT. A surprising omission from consideration so far is HOS [HAMI76], which has a strong formal basis, and would seem to have taken off commercially recently.

5.3 Task 3: Development Language

The overall long-term objective of Task 3 is to design a prototype, experimental environmental notational system for expressing software design and development methods. The first step towards that goal, Task 3.1, is to get a feel for the state-of-the-art. There has been little attention paid to design and development methods as objects of concern in their own right. Since there has been so little attention paid to development methods as objects, it is no surprise to find that even less attention has been given to a language which would express these methods in a formal manner.

Task 3.1 concentrated on a small group of projects, considered interesting for a number of reasons:
these should be well-documented, accessible, quality projects, exemplifying a major approach towards expressing design knowledge; the range of projects should be at the forefront of knowledge in industry, development and research, vid [PART83].

The projects looked at were:
- Darlington's transformational system [DARL82] (Imperial College London); It builds upon the Burstall-Darlington rules for unfolding and folding.

- Paddle-Popart [WILE83] (D. Wile, ISI, Los Angeles); this is a working, complete Interlisp-based environment, which supports a transformational approach in which composition methods are introduced explicitly.

- CIP [MOLL83] (F. Bauer's group at the Technical University, Munich); This project, built on solid semantic foundations, covers the range from logico-algebraic specifications down to machine-level programs; the design knowledge is expressed as conditional, parameterised rewriting rules. It has been applied to the design of the support system itself.

- SPRAC (CERT, Toulouse); This is an example of the database approach. Design information is kept and controlled at the structural, syntactic level; linguistic frameworks are provided for specifications, programs, and developments. SPRAC exemplifies practical software engineering environments developed in research.

- Attribute grammars in compiler design [KAST82] (University of Karlsruhe and GMD-Karlsruhe); the compiler design area is of interest because of availability of deep design knowledge and powerful tools. Attribute grammars constitute an important means of expression because they are akin to abstract data types and logic programs, and also because they are executable efficiently.

- PROMOD (Industrial product - GEI, Aachen, Germany); this system integrates the full life-cycle, from SADT-like techniques of requirements analysis to software modules. It represents the current state of the art in industry.

Complementary Approaches
The study of complementary approaches to those already studied (e.g. knowledge-based approach) is starting now.

Some conclusions are:
- means for formally expressing development methods are not available.
- specifications arising from nonfunctional requirements are neither expressed well nor well-integrated with other specifications.

Task 3.2 (the definition of the requirements for the development language) has been amalgamated with the continuation of Task 3.1. The topics to be investigated in the immediate future are:
- Knowledge-based systems
- Conceptual modelling in the context of databases
- Functional, logic, or relational languages for specifications and programs, e.g. FP, ML, EQLOG, or Larch [ESPR86].
- Constructive logic for the semantics of design knowledge.
These studies will crystallise the properties which a development language should ideally have, and also give a feel for which properties are feasible.

5.4 Task 4: Support Environment Synthesis

The main objective of Task 4 is the production of a prototype environment for DEVA so as to allow experimentation with methods and strategies. As already stated, Task 4.1 (evaluation of existing environments) was closely associated with Task 1.1 because studying environments was inevitable while studying tools.

The work has mainly focussed on the integrational aspects of an with respect to its tools. Classification schemes for tools and environments have been proposed and applied to a wide selection of tools and environments including Unix, PCTE, PROMOD [HRUS82], SEEK [EPSI84], GRASPIN, CHI, AFFIRM [GERH80], RAPID, Tektronix/SA, PAPS, ARCTURUS, Interlisp, Microprolog, SPRAC, OASIS, ADELE, Mentor, SPBS, Zetalisp, Programmer's Apprentice, Toolpack [ESPR86].

In line with the general objective, a framework for the environment should be designed, so Task 4 focusses on the GLOBAL aspects of environments rather than the particular i.e. what is the architecture of an environment, what are the links between tools, their interfaces, how integrated are they, how difficult is it to add new tools etc. More specifically the objectives of Task 4.1 are:
- a global examination of existing environments according to some fundamental keywords (ARCHITECTURE, METHODS supported, INTEGRATION of tools or concepts making up the environment, APPLICATION classes, TARGET systems).

- a deep understanding of the functionalities needed to support the strategies or methods expressed with DEVA This understanding will facilitate the preparation of the next subtask.

Results
The first results have shown the lack of methods for configuring environments and for supporting methods of software development.

Future activities will include:
- Extracting from existing environments the concepts most useful for the support environment
- A reconcentration on the study of AVAILABLE existing environments i.e. SPRAC, AFFIRM, SEEK-MENTOR, all hopefully available on Unix
- the studies of sophisticated environments will continue (e.g. CHI, KEE) but the studies will be focussed on specific points, for instance, Knowledge Engineering.

5.5 Task 5: Application Case Studies

Task 5 is concerned with two activities. The first one is related to the application domain for case studies (T5.1,T5.2), and the second one centres on the acquisition and manipulation of software design knowledge.
Task 5.1 (Selection) evaluates a number of possible case studies with respect to how well they represent real world software development problems and experience. Particular emphasis is to be placed on the relationship to tasks 1.4 and 2.3 in evaluating tools, environments and methods. The planned outcome is a description of representative case studies for evaluation by other tasks.
Task 5.3 (Knowledge acquisition and representation) selects the knowledge from specific application areas and suggests specific notations to represent the knowledge.
The delays in the formal signing of the ToolUse contract by the CEC have delayed work on this Task. To make up some lost time two initially small case studies with scope for considerable increase in complexity have been started. These two case studies were selected to provide early feedback by experimentation:

- a message passing system
- "the" package router problem

The specification of functional and non-functional requirements has initially been kept simple to allow an early start to experimentation within the limits of available manpower. These small problems permit the study of how the addition of further requirements, especially non-functional requirements, affect the design. Two other case studies are a LIFT study and a PUBLISHER system; both have been formally verified. Also in the next 6 months, the requirements for small experiments will be formalized using SARS [EPPL83].

Several large industrial case studies will be studied. One involves a Bank with 350 branch offices, which has written a banking system consisting of 500,000 lines of source code which took over 100 manyears to develop. The implementation of this system has just been completed. This system, together with documentation, design history and decisions, will be available for study and access will be provided to relevant staff. Another possibility is a high-security real-time command control and communications system. A draft study scheme for proceeding with case studies has been prepared.

In the medium-term this Task is interested in how the application area characteristics have an affect on design and development decisions, and consequently in identifying parameters for describing application areas. A preliminary classification scheme is by:

a. functions and data
b. typical requirements
c. life cycle model

Task 5.3 is concerned with developing a knowledge acquisition model for capturing requirements. This work has just started. A preliminary classification of requirements is:

a. functional versus non-functional
b. application system requirements vs target system characteristics
c. Application area classification of requirements:
(1) e.g. Realtime, ...,Data Processing
(2) Collection of typical non-functional requirements based on experience.

5.6 Task 6: Target Systems

The main objective of this task is to facilitate the reuse of existing components of target systems. Therefore it is necessary to describe the functional and operational properties exhibited by the hardware and software of target systems. The goals of Task 6 are:
- the derivation of models for specific target systems
- the integration of these models within the framework of the development language DEVA.

Not a lot of work has yet been done. It seems to be difficult to describe a useful formal model in this area, other than the conventional hierarchical layer model of application software, basic software, and hardware on the bottom, e.g. Libraries of a MODULA-2 system; MODULA-2 system; IBM-PC. The mid-term responses from this task will be:
- exploring the role of such target system descriptions in a SEE
- Interaction with DEVA: a linguistic framework is necessary to work with target system descriptions; propagation aspects.
- Compile results from other research groups on the propagation of non-functional requirements and software component reusability.

The long term actions will be:
- Investigation of formal methods to describe computer systems
- Development of a formal framework to describe target systems.

5.7 Task 7: Logistic Support And Management

The scope of Task 7 encompasses all the traditional project management activities with their logistic support, and in addition the peculiarities of such a project:
- an R and D framework with very ambitious objectives;
- wide distribution of partners both geographically and organizationally;
- highly correlated tasks in the original workplan;
- specialised expertise and interests among partners;
- communication in a multi-lingual, multi-organizational, multi-national project.

The project organization which was set up includes a project manager, with an assistant responsible for logistic support, a technical board for arbitration between partners and task coordination according to the project objectives.

Such a structure with a control body and distributed sites requires that exchange of documents be actively promoted and that a common understanding of the project objectives be reached by team leaders within a short period of time. This has been encouraged by a document referencing scheme, by a classification scheme for the project literature base, by project organization guidelines and by the use of Eurokom, a computer-based conferencing system.
Task coordination and monitoring is aided by:
- Project status is described at regular intervals.
- Assistance is given to late starting tasks
- Reuse of existing knowledge is promoted.
- Technical fora for promoting exchange of ideas on a six-monthly basis.

6.0 GENERAL COMMENTS

This discussion is based on one team member's perception of the project. The TCD team of which he is a member

concentrated on Tasks 1 and 4 in this time, so the coverage of other areas could be deficient. A description and commentary on the project which would be more balanced and more in agreement with the perceptions of the project manager and the technical board will be given at the Esprit '85 Review Week in Brussels in late September 1985 [ESPRIT86]. It should be noted that barely nine months have passed since the contract was signed.

The research objectives are very ambitious and technically daunting. Obviously it is hoped that the benefits of cross-fertilization of ideas will greatly outweigh the disadvantages. There are obvious goals and benefits to the CEC of trying to foster cooperation across national boundaries as well as cooperation between universities and industry.
At this stage we can say that cooperation is working; that the teams are working in general reasonably well on their tasks, but that cooperation and communication should be improved further.

There is a general feeling at this stage and at the lowest level in the hierarchy of a considerable vagueness in what the project is trying to do. This could be improved by an overview of the project; this will come out of the Esprit 85 Review and also out of exercises like this one. Perhaps more effort should be put into communicating with the grass roots, as insights and inspiration coming from this level may be more valuable in not being bound by the prejudices of experience - "where angels fear to tread .....". To this end perhaps more use should be made of short visits by workers to other teams' sites to encourage cross-fertilization of ideas.

People have been concentrating on their tasks to a great extent, and not making sufficient effort to communicate with other teams. In TCD, it is acknowledged that we have received requests for critiques of work to which we have not responded. Perhaps this will improve with some effort in the near future. A considerable amount of the attention of all workers in the first year has been spent in just getting the show on the road; other items with lower priority have been ignored. There has been a tendency for teams to work too much in isolation and to consequently get carried away with their own work without reference to project goals. An example is with Tasks 1 and 4, where the environmental classification scheme seemed to take on a life of its own.

There should be significantly more emphasis on experimentation. Classification schemes on their own are boring and generally uninteresting, even if they are essential for comparative purposes and for structuring the work. Experimentation gives tangibility with consequent benefits in insight and inspiration.

7.0 SUMMARY

ToolUse will have been in action for a year officially on October 15, 1985. In this year, some solid achievements have been made:
- The project has been got on the road; teams are in general working well on their tasks; and cooperating with perhaps some improvements to come. Because of the very high connectivity between project tasks cooperation will be much more necessary in the future.
- Multi-team, multi-national, multi-lingual, multi-cultural research works; but the next open question is will the quality of the end-product be high?
- The crunch-point in the project will come in the next eighteen months to two years and is likely to revolve around the point of whether a formal methodology can be put into a framework suitable for industrial large scale products.

A personal opinon is that the project is on the right track; that software engineering is an engineering discipline which requires the right mix of semi-formal and formal techniques (as electronic engineering for instance); that a knowledge-base approach is necessary; that studies into why industry is reasonably happy with its existing techniques will be useful; that inferential programming approach indicates a useful direction to take; that reuse of programs will be extremely useful.

Industry has a considerable inventory of software products. These products are constantly being modified and updated. An automatic method that would help solve this problem alone would be a major advance. Towards this end, it seems that industry would like the "objects" for manipulation to be up to module size say 2 to 16K or so. From a search of the literature, it appears that this sort of object would give severe, if not fatal difficulties, to any of the current methodologies. In addition the vast majority of industrial software has not been formally verified. Inferential programming offers in theory the ability of working back from an implementation so as to generate a specification for an existing working program; industry might prefer however to take an existing program and transform it to another while still maintaining the credibility factor of the original program. At this point in time, it would appear that while these changes can be made relatively easily manually, a system that could do it automatically is some time away.

The major problem areas that I see for ToolUse are:
- what transformations should be used, including how many and how big
- what size objects should be manipulated; ideally they should cover a wide range.
- what knowledge representation should be used?
- what formal notation is suitable in view of the above?
- all these questions are obviously interlinked

The major benefits from ToolUse, at least, will be an ARTICULATION of the problems of treating design methods as objects; this may in turn give rise to significant SIMPLIFICATIONS in design methods, rules, transformations, tactics and strategies.

RESEARCH WORKERS:
S Baker, M Dausmann, O Declerfayt, P de Groot, J L Durieux, D Dzierzgowski, H Gibbons, J F Gleeson, H Horgen, A Hunot, R Jacquart, S Jaehnichen, D Kautzmann, S Kearney, G Koch, H-J Kugler, M Lemaitre, M Lemoine, P Luchner, B Mathews, P Moran, D O'Neill, M Pasa, J A Redmond, K T Ryan, K Singer, M Sintzoff, K Wachsmuth.

I would like to thank M Brady, S Kearney, D J O'Neill and K T Ryan for comments and discussions. I would especially like to thank H-J Kugler for many discussions and for comments on a draft of this paper.

Postscript: The most visible OBJECT in the view from the trenches is a large paper mountain.

References

[BALZ81] R. Balzer "Transformational implementation: an example" IEEE Trans. on Software Eng. Vol SE-7, No 1 Jan 1981 pp 3-14.

[BOEHM76] B. W. Boehm "Software Engineering" IEEE Trans. Comp. Vol C25, No 12 Dec 1976 pp 1226-1241.

[BORG85] A. Borgida, S. Greenspan, and J. Mylopoulos "Knowledge representation as the basis for requirements specifications" Computer April 1985 pp 82-91.

[BOUR85] J. P. Bourguignon "PCTE: a basis for a Portable Common Tool Environment" in ESPRIT '84 ed. by J Roukens and J. F. Renuart CEC Brussels Belgium North Holland 1985 pp 75-84.

[DARL82] J. Darlington "Program transformations" in Functional Programming and its Applications. eds Henderson, Turner Cambridge University Press 1982.

[DONZ80] V. Donzeau-Gouge, G. Huet, G. Kahn and B. Lang Programming environments based on structured editors: The MENTOR experience. Res. Rep. 26, INRIA Le Chesnay, France 1980.

[EPPL83] W. K. Epple and G. R. Koch "SARS - a system for application oriented requirements specification" IFAC Real Time Programming pp 43-50 Hatfield UK 1983.

[EPSI84] Epsilon GmbH "SEEK - software engineering environment kernel", Reference Manual/Draft, Version 0.2 (internal), 1984

[ESPR86] H Horgen "Esprit Project 510: ToolUse" in ESPRIT '85: Status Report of ongoing work to be published April 1986 North-Holland.

[FREN85] K. A. Frenkel "Towards automating the software-development cycle" Comm ACM Vol 28, No 6 June 1985 pp 578-589.

[GERH80] S. Gerhart, D. R. Musser, D. H. Thompson, D. A. Baker, R. L. Bates, R. W. Erickson, R. L. London, D. G. Taylor and D. S. Wile "An overview of Affirm: a specification and verification system" pp 343-347 in Information Processing '80. ed S H Lavington North-Holland New York 1980.

[GREE82] C. Green, J. Philips, S.Westfold, T. Pressburger, B. Kedzierski, S. Angebranndt, B. Mont-Reynaud and S. Tappel Research on knowledge-based programming and algorithm design - 1981" Rep. Kes. U. 81.2 Kestrel Institute, Palo Alto, California 1982.

[GUTT82] J. Guttag, J. Horning and J. Wing "Some notes on putting formal specifications to productive use" Science of Computer Programming Vol 2 No 1 October 1982.

[HAMI76] M. Hamilton and S. Zeldin "Higher order software - a methodology for defining software" IEEE Trans on Software Eng. Vol SE-2 No 1 March 1976 pp 9-32.

[HEYD85] P. Heyderhoff "GRASPIN: A coherent methodology" J. P. Renuart CEC Brussels Belgium North Holland 1985 pp 75-84.

[HRUS82] P. Hruska "PROMOD: Motivation and Introduction" Version 1.2 GEI Aachen, F R Germany July 1982.

[JACK83] M. A. Jackson "System Development" Prentice Hall 1983.

[JONE79] C. B. Jones "Software development - a rigorous approach" Prentice Hall 1979.

[KAST82] U. Kastens, B. Hutt and E. Zimmermann "GAG: a practical compiler generator" Lecture Notes in Computer Science 141 Springer 1982.

[LEMA83] M. Lemaitre, M. Lemoine, and G. Zanon "SPRAC: a computer assisted softwware development system" in 'Tools and Notions for Program Construction' ed. D. Neel Cambridge University Press 1983.

[MOLL83] B. Moller, H. Partsch and P. Pepper "Programming with transformations: an overview of the Munich CIP project" submitted for publication 1983.

[PART83] H. Partsch and R. Steinbruggen "Program transformation systems" Computing Surveys Vol 15 No 3 pp 199-236 Sept 1983.

[SCHE83] W. L. Scherlis and D. S. Scott " First steps towards inferential programming" IFIP 83 pp 199-212 Elsevier 1983.

[SILV85] B. G. Silverman "Software cost and productivity improvement: an analogical view" Computer Vol 16, No 5 pp 86-96.

[TURS84] W. M. Turski "Completeness and executability of specifications" in Proceedings of Software Process Workshop Egham, U K February 1984 IEEE pp 153-156.

[WILE83] D. S. Wile " Program developments: formal explanations of implementations" Comm ACM Vol 26 No 11 Nov 1983 pp 902-911.

EXPERT SYSTEMS – IN COMPUTER AND ELECTRONIC SYSTEMS.

James A. Redmond & John F.J. Gleeson.

(Dept. Of Computer Science, Trinity College, Dublin, Ireland)

UDK: 681.3:159.953

ABSTRACT.

An outline of an approach to the use of expert systems in fault diagnosis of electronic circuit boards is given. Lessons are drawn from existing successful expert systems such as R1 and Steamer. The Esprit project 96 is described with some comments on progress to date. Some guidelines and future directions are given.

1.0 INTRODUCTION.

Considerable interest has been shown in so-called Expert Systems in the last few years; an introduction is given in [NAU83]. This has been due to the success of some applications, changes in technology - hardware and software, and decrease in the costs of exploiting the technology. Figure 1 shows the dramatic decrease in the time to implement a system. Table 1 (appendix) shows the wide range of Expert System Shells available, and their wide range in cost. Figure 2 shows an estimated demand curve for "Knowledge Engineers" in the near future [JOHN83].

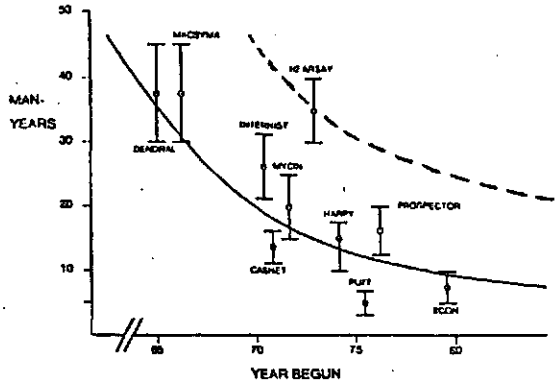


Fig 1.

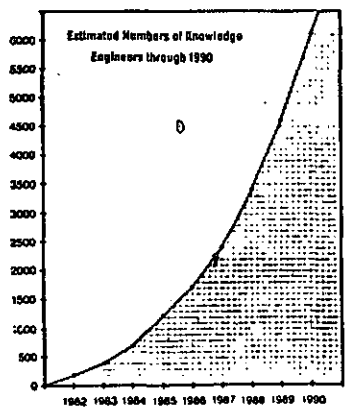


Fig 2.

The dramatic decrease in the number of manyears to develop an expert system is due to a number of effects:

- The well-known learning curve effect.
- The growing expertise in building expert systems.
- People are better at picking suitable application areas.
- Better software and hardware tools are available cheaper.

What is an Expert System? There is considerable scepticism in some circles at this question; some cynical responses have included "Any Lisp program" to "A user interface to an application program". However while we may argue about the definition and its usefulness/appropriateness, there is no doubt that some expert systems applications have been successful e.g. Mycin, R1, Dipmeter [FEIGE81] [MCDER82] [MCDER84].

What are the characteristics of an Expert which are of interest for modelling? Experts can:

- Apply their expertise in an efficient manner. They can employ plausible inference and reason from incomplete and uncertain data.
- Explain and justify what they can do.
- Communicate well with other experts.
- Restructure and reorganize knowledge.
- Break rules. They understand both the spirit and the letter of a rule. They have many exceptions.
- Determine relevance. They know when a problem is outside their expertise, and when to make referrals and to whom.
- Degrade gracefully in their problem-solving performance at the limits of their expertise.

Present-day expert systems have modelled the first three of these, and research has started on Explanation and Knowledge Acquisition [MILSN85].

2.0 EXPERT SYSTEMS

Expert systems have developed gradually from early research into AI and they can be described as computer programs which perform certain tasks, traditionally regarded as requiring human expertise or intelligence, at a comparable level to the human expert. The tasks which experts perform are usually seen as requiring intelligence and such tasks or operations are often extremely difficult, or impossible to describe in algorithmic form. This is because experts do not work to a rigid plan, but apply experience and judgement to decide how the task should be performed. Throughout an operation the expert will make decisions about how the operation or task should proceed, at any stage he may only be able to see a few steps ahead, and may have no idea about how he will complete the task until the very end. This is in contrast to conventional programming, which has a definite algorithm and a clear path to the completion of the task. Even where conventional programs have complex branches, loops or recursion, the underlying algorithm can be expressed in a definite manner.

Experts generally use a collection of heuristics to perform tasks, solve problems or control operations. These are informal judgemental rules or "rules of thumb" [FEIGE81]. Such rules are often very imprecise. Imitating or modelling these tasks is therefore extremely difficult in a procedural type language, for example Fortran or Pascal. If any kind of modification is required to a vague rule , as the expert learns more about the problem, then this could result in major alterations to the program. For these reasons conventional software approaches to such tasks have not succeeded. Recently however, the development of Expert Systems has meant an enormous reduction in the complexity of computer programs which exhibit this kind of "intelligence" or reasoning.

A number of very successful expert systems have been developed for a variety of applications. The PROSPECTOR system [FEIGE81], developed for mineral exploration, recently discovered deposits of minerals estimated to be worth about $100 million [MICHE79]. MYCIN [FEIGE81] [SHORT76], a medical diagnosis system for treatment of blood and meningitis infections, and the basis for EMYCIN [HAYES83] [VANML81], has outperformed medical experts. Digital Equipment Company (DEC) use a system called XCON (the commercial version of the system called R1) [MCDER82] [MCDER84] to configure the VAX range of computers. Among the other diverse applications for existing expert systems is a system for Mass Spectroscopy analysis called DENDRAL [FEIGE81] and a system Schlumberger are developing for analysing data associated with oil wells [COX84].

Special reference will be made in this paper to a system called Processor. This is an EMYCIN based expert system, which carries out electronic fault diagnosis on M6809E microprocessor systems. This system was developed by one of the authors.

It can be argued whether or not these systems actually exhibit any real intelligence. Regardless of this their potential power and value can not be questioned. Expert systems of the type described above are going to become increasingly common in the future, as more commercial systems are developed.

As the Japanese Fifth Generation computer project [STONE84] [[FEIGE83b] proceeds, expert systems are expected to become increasingly important within that project, and also in related or competing research programs. As a result, expert system methodologies will have an increasing influence on the development of new software. This influence is expected to be felt across the computer industry from microcomputers to mainframe computers [COX84] [FREN85].

2.1 OUTLINE STRUCTURE OF AN EXPERT SYSTEM

Expert systems are a class of Artificial Intelligence (AI) programs, and their structure is similar to that of AI programs generally (see Fig 3). That is they consist of three parts defined as follows:

1: A Database.

2: A Knowledge Source.

3: An Inference Engine.

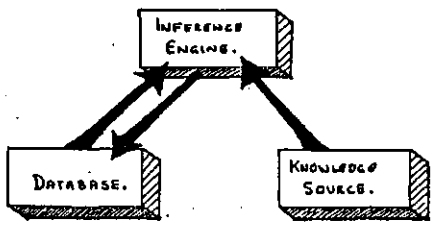


Fig 3

2.2 THE DATABASE

All the programs variables or facts, would normally be contained in the database. It is the area of memory used to store these along with the necessary procedures to store and retrieve the relevant data or facts. Initially the database would contain the facts or data which the expert system uses to derive or infer high level information. For example a patients medical history may be stored in the database, before the expert system attempts to reach or infer a diagnosis or treatment. The contents of the database are also used to answer questions about the current consultation.

2.3 THE KNOWLEDGE SOURCE

The knowledge source is made up of information or 'knowledge' provided by the expert, and represents his 'expertise' in the given area. Normally such knowledge will be heuristical and therefore some well defined format must be adopted to represent these heuristics.

A domain expert's heuristics are normally similar to 'rules of thumb' and often experts apply combinations of rules in a 'rule of thumb' manner, based on insight or experience. This has been useful to expert system designers, and many of the more successful systems are rule based, for example MYCIN.

The easiest way of encoding these 'rules of thumb' was found to be in the form of simple low level production rules, for example:

IF <PREMISE> THEN <ACTION>

Although each production rule is individually simplistic, collectively they can represent a large number of complex and interacting ideas, representing a considerable amount of experience.

In an expert system for computer aided design of electronic circuits [COX84] [STEFK83], a typical rule might be that pin number 9 of of a 6116, 2716...IC should be connected to dataline D0, and this would be expressed in production rule form as:

IF <IC IS A <6116 OR 2716 OR....>> AND
<CONNECTING PIN 9>
THEN <CONNECT TO DATA LINE D0>

Expert Systems with some sort of framework [FEIGE81], are generally domain independent, until the domain specific knowledge has been incorporated into them. To change the problem domain, all that is required is to change the incorporated knowledge, ie the rules. This is the principle of EMYCIN, which is an expert system builder, developed from MYCIN (EMYCIN stands for Empty MYCIN). The Processor system was developed by adding rules to the EMYCIN system. Processor currently contains 213 rules. If these production rules, or what ever other data structures are being used, are regarded as pieces of knowledge, then the addition of new rules or data structures is like adding knowledge to the expert system. An attribute or characteristic used to describe this aspect of a particular representation schema, is modularity. Modularity refers to the ability to add, modify or delete individual production rules or data structures, more or less independently of the rest of the database of rules/structures.

Humans often find modular or clearly divisible systems easier to understand or work with, especially once a system becomes large and the number of interactions increases. In such cases perceiving the total system with all its interactions is virtually impossible, and it becomes essential to adopt a modular view of the system. However a great deal of human knowledge is inherently non-modular, and it is often extremely difficult to extract from an expert and then to express in rule format or in a definite modular fashion.

2.4 CERTAINTY FACTORS.

Rules in many systems including the MYCIN or EMYCIN systems are described as being judgemental [FEIGE81] [SHORT76] [VANML81], this implies that they make, or are used to make, inexact inferences on a confidence scale ranging from -1 to +1, where +1 represents complete confidence in the proposition, and -1 represents no confidence in the proposition, or complete confidence that the proposition is false.

When the MYCIN system was being developed, attempts were made to incorporate formal statistical measures, however these were rejected, partly due to the implementation difficulties of incorporating them, but more importantly, because it was found to be very difficult to reflect the medical experts selection or weighting process in conventional or standard statistical frameworks. Doctors did not appear to use anything resembling a statistical approach when deciding between alternatives, they weighted the choices as strong or weak, so a straight forward weighting scale was adopted for the certainty factors in MYCIN.

In these systems, a certainty factor may be regarded as a measure of the strength of association between premise clause and action clause of a given rule. When a particular rule succeeds, it is because its premise clauses are true in the current context of the system, however the certainty factors of the component clauses of the premise are combined to form a new overall certainty factor for the premise. This premise certainty factor is then used to modify the certainty factor of the resulting action clause. Therefore, if a premise succeeds, but with a low combined certainty factor, in other words with a low degree of confidence, then the resulting actions taken are assigned a low certainty factor also.

2.4.1 CERTAINTY FACTORS IN THE PROCESSOR SYSTEM: -

Certainty factors, which are an important feature of the EMYCIN system, were used very little in developing the Processor expert system. The reasons for this are very interesting, and are a reflection of the problem or fault domain. In most instances the system asks the user if there is a signal, pulse or activity at a given point on the board, this is a simple yes/no query; there is very little vagueness, or ambiguity, as there is a pulse or there is not! If the expected pulse or signal is not there, then there is a fault or problem in the circuit which produces or uses that signal or pulse, and that is definite. So the system localises to that part of the circuit, and tests another point, or section of the circuit; again it either finds what it expects to find or it does not. There is very little room for uncertainty in digital electronics, a line is generally either high, low or carrying a signal! The nature of digital test equipment also leads to precise results to tests, you either see a signal on an oscilloscope screen, or you don't. This is very different to the situation in medical diagnosis where estimates and qualitative judgements must be made without clear quantitative results.

It can be concluded therefore that there is very little need or possible application for certainty factors in systems of this type and this is a reflection on the fault or problem domain of digital electronics.

When certainty factors are used to any great extent in a system, they decrease the predictability of the system, as the tree-like structure of possible consultation paths becomes much more complex and vague, and therefore more difficult to predict. This has the effect of increasing the mystique of the system, and thus the system appears to be more "intelligent" to the user. The complexity however, or unpredictability of an expert system, is not a measure of the "intelligence" of the expert system, and this is an extremely important point. In fact increased use of certainty factors, however invaluable in certain circumstances, implies decreased certainty in the result of a consultation. Perhaps they would be better called "uncertainty" factors.

2.5 THE INFERENCE ENGINE

The purpose of the inference engine is to take the collection of rules which form the knowledge source, and the initial database of facts, select an applicable rule and apply it. The result of applying a rule modifies or updates the database. This operation is repeated until the final goal state or solution is reached.

When the inference process is data driven, the premise of each rule would be checked against the contents of the database, to find the applicable rules from the rule set. One of these rules is then selected and applied, the action part of the rule, once applied updates the database creating a new sub-set of rules which are applicable. The inference engine can select the rule to apply either by taking the first rule it finds which can be applied, by selecting the rule which appears to have the shortest path to the goal state or alternatively the rules may be assigned priorities by the builder of the system, so that the rule with the highest priority will be selected. The second of these options is often extremely difficult to incorporate into a system. Assigning the rules priorities adds another dimension of complexity to the system, increasing the argument in such cases for a modular approach. Most systems, however, simply select the first applicable rule.

A data driven procedure may be illustrated as follows: Ref: [COX84]

```
Database Is Initialised;
Repeat Until Database Satisfies Goalstate
    Compare The Condition Part Of Each Rule
    With The Contents Of The Database;
    Select A Rule From The Applicable Rules;
    Apply The Selected Rule, Thereby Updating
    The Database;
End;
```

In a medical expert system, the goalstate might be a diagnosis, treatment or therapy, for example.

Goal driven systems represent a top down strategy whereby the rules are chained together so that the output from the <action> part of one rule, is used in the <premise> part of a later rule to establish its applicability, or eventually to satisfy the goal state.

In many cases the nature of the knowledge source will dictate, or at least influence, what control strategy should be adopted. For example an expert system for electronic fault diagnosis, might be goal driven, that means it might expect a 14 MHz signal at a particular pin of a given IC, and would look to see if this signal was there. Alternatively the system could be data driven where it looks at what signals are at the various pins and does analysis of the result to determine what is actually happening on the board.

Where possible a combination of data driven and goal driven strategies within a single system would produce the best and most elegant system. Whatever approach is adopted the control strategy of the inference engine could not be regarded as being responsible for the seemingly intelligent behaviour of an expert system. Feigenbaum [FEIGE79] makes this point:

> 'The power of an expert system derives from the knowledge it possesses, not from the particular formalisms and inference schema it employs.'

With this in mind it is interesting to note that expert systems are often referred to as 'knowledge based systems'.

2.6 EXPLANATION FEATURES OF AN EXPERT SYSTEM

Although the basic building blocks of virtually all expert system programs, are the database, the knowledge base and the inference engine, various other facilities are often desirable. Of the extra facilities incorporated, explanatory facilities are possibly the most important. The ability to ask for an explanation of a particular result, or why a certain question is being asked, inspires confidence in the users of the the system, as it allows the users to judge the results of a consultation for themselves. The users are given the feeling that they are getting an opinion from the system, not being ordered or told what to do by the system. This is very important in applications where computers were not previously employed, as resentment and suspicion of such systems is often common. Like any computer system, an expert system, has to be liked by its users if it is to be successful. Therefore systems should be as user friendly as possible. In many cases simply phrasing the questions and results clearly in good natural English will make a noticeable difference to the user's view of the system. Where possible replies from the user, should be accepted in natural or near natural language. Alternatively replies from the user could be through menu tables or some suitable graphical input facility (such as touch screen or mouse).

From the point of view of inspiring confidence in the system, by far the most important feature to be included is the explanation facility, and it is questionable whether this should be regarded as an essential part of an expert system program, like the knowledge base, or merely as an additional feature. Without an explanation facility, the 'intelligence' of the system is hidden from the end users, and also, to a certain extent, its expertise.

An example of the use of the HOW and WHY facilities in the Processor system, is shown below:

```
.
.
.
7) Is the Vcc input to the EPROM (pin 24) connected to
   the +5 Volt line? (YES/NO):
** WHY

[ie WHY is it important to determine whether the Vcc
    pin of the EPROM is connected to 5 volts?]

[2.0] this will aid in determining whether the memory
     mapping decode circuitry will be examined.
     It has already been established that
       [2.1] The EPROM is the suspected fault area, and
       [2.2] You are certain that the EPROM test
             program was loaded correctly into the
             EPROM, and that the EPROM IC was inserted
             correctly into the socket, and
       [2.3] The EPROM ground pin is connected correctly
             to ground
     Therefore, If
       [2.4] The Vcc pin of the EPROM is connected
             correctly to 5 volts, and
       [2.5] The Vpp input of the EPROM IC has been wired
             correctly, and
       [2.6] With reset pressed the EPROM is not being
             selected
     Then
        It is definite (1.0) that the memory mapping decode
        circuitry will be examined.
      [RULE061]

   [Back to question 7...]
** HOW 2.5

[ie HOW is it determined whether the Vpp input of the
    EPROM IC has been wired correctly?]

     There are no rules which allow the deduction of
     whether the Vpp input of the EPROM IC has been wired
     correctly, so a question will have to be asked.

   [Back to question 7...]
** YES
8).....
```

The query mechanism provided by EMYCIN is virtually identical to the facilities in the MYCIN system [VANML81]. When asked a question at any time the user may respond "WHY", if the response does not satisfy the user, then they can ask "HOW" a particular part of the explanation provided was arrived at. The system replies by listing the relevant rule, but not in its Lisp form, instead it lists the properties of each parameter of the rule with the desired value that they should have for the rule to succeed.

Processor is a production rule system with 213 rules implemented in EMYCIN on a DEC-20 in TCD by J. Gleeson. It does electronic fault diagnosis on an M6809 microprocessor card. This project provided experimental input to Esprit Project 96 (discussed later).

If a sophisticated explanation facility is required, it may be best implemented by secondary expert system, giving rise to what is termed, a multiple expert system.

Table 1 shows some sample expert systems developed in TCD. All these systems are production rule systems , all are experimental, some have led to on-going work and potential future development, eg Processor, Floors, Lending. One, MRCPI, has been quite successful and it is planned to use it in its present state.

In the remainder of the paper we will look at some examples of expert systems. R1 is an example of a first generation expert system. Steamer is arguably not a full second generation expert system. Project 96 is an example of a design for a full second generation expert system.

3.0 TWO SYSTEMS OFFERING USEFUL INSIGHT.

3.1 R1: The Evolution Of A Production System.

R1 (now known as XCON) [MCDER80] [MCDER84] is a program that configures VAX computer systems, originally developed by McDermott at Carnegie-Mellon for Digital Equipment Company It is a production system. 50 to 150 components make up a Vax computer system. R1 has sufficient knowledge of the configuration domain and of the peculiarities of the various configuration constraints that at each step in the configuration process it simply recognizes what to do. Consequently little search is required in order for it to configure a computer system.

The history of R1 gives some useful insights for the development of expert systems, especially production systems. R1 was originally written in OPS4. In the first four months, 200 rules were written by extracting knowledge from experts in the absence of a system. In the next 4 months, 600 rules were generated. These rules were generated by omissions not covered in the first 200 rules. In general the omissions were of items which were "obvious" to an expert.

R1 is considered by DEC to be a successful expert system; it has been in use for over 5 years, and has configured over 70,000 computer systems. One in a thousand misconfigurations occurs. At present seventy people support the R1 system. R1 has been significantly expanded in the five years to cover additions to the VAX range as well as to the PDP-11 range. It is possible that the problem solved by R1 could be formulated as a knapsacf-style problem, and solved by dynamic programming. However it is possible that it would be combinatorialy bound. If solvible a table of solutions for the various pieces of equipment required would be available with significant cost and time savings. Another advantage would be that the cost function would be minimized in arriving at a solution. A criticism of R1 is that while a feasible solution is usually derived, it is not known if it is close to the minimal cost for the configuration.

The originators of R1 considered that at some stage R1 would be finished. It is interesting to note that it

is still not finished and continues to grow. The growth in the number of rule with time is given in Fig. 4:

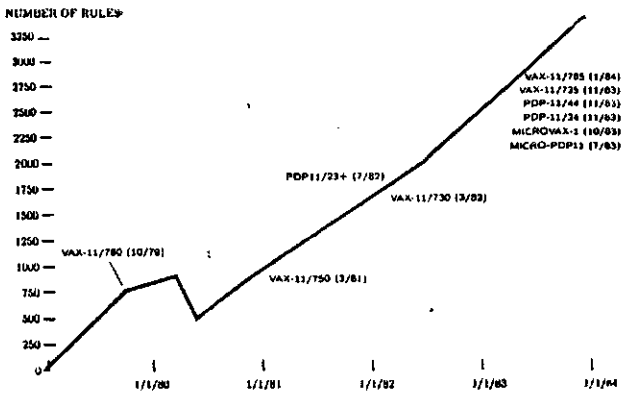


Fig 4: Number of rules in R1 over time.

The growth in manpower has been from 5 people originally to 8 after 2 years to 77 after 5 years.

A final comment by McDermott:
"The real world treats AI tools with the same disrespect as it treats all other tools".

3.2 STEAMER: State Of The Art Explanation Facilities.

Frame and network-based approaches allow the implementation of "deeper-level" reasoning such as abstraction and analogy - important expert activities. Also object-oriented programming can be used to organize collections of rules. In object oriented programming, objects can be given behaviours, and therefore the control of rules can be distributed into rule, rule-packet and domain objects. For example, we could represent objects (e.g. a PUMP) and the processes (e.g. "start instructions") for it. Frame systems also provide a system to inherit attributes from a taxonomy of entities. The control of frame or semantic-net systems is usually much more involved than surface systems and they are usually implemented in a way that an explanation facility cannot get at. Steamer is a training aid developed for the US Navy for the purposes of training personnel in the operation of shipboard steam plants. The operating procedures consist of a series of steps or subcomponents of the plant. The components and procedures are represented as frames in Steamer, as are the abstractions of components and procedures that experts use in teaching steam-plant operation. The steps of a procedure come from the observation of the components (and subcomponents) of the device to which the procedure applies. The ordering of the steps comes from a third represented entity - operating principles, which are culled from experienced operators and represent "compiled" or "distilled" knowledge of steamplant operation.

Steamer is an example of a multi-level system (i.e. a system that uses knowledge represented in different forms). Steamer uses the following representations:

- An ICONIC representation of the objects of the steam plant e.g. valves, pumps, tanks, dials etc.
- A FRAME representation of steamer objects, procedures and operating principles; used for describing, explaining, categorizing, abstracting and referencing.
- An ASSERTIONAL database where assertions about steamer can be made and retracted
- A quantitative simulation used for observing cause and effect of the application of processes.

Active value methods, which are used, are found to be extremely useful in that once a numeric value is changed all the other representations are adjusted accordingly.

The underlying concepts of Steamer have been reflected in other projects. The basic underlying, motivating ideas in Steamer are:

- The Conceptual Models which an expert uses need to be understood for constructing useful applications.
- Graphical Interfaces for INTERACTIVE, INSPECTABLE Simulation.
- Conceptual Fidelity: model the conceptual models
- Implementation Philosophy: Develop a useful system concerned with a real problem in a complex training domain making fruitful use of abstraction.

A detailed simulation for a shipboard steamplant already existed before Steamer was built. A colour graphics interface was considered a sine qua non so that one could view and manipulate the plant at a number of hierarchical levels (100 colour views are available). Knowledge representation was necessary for adopting various perspectives of the plant and for maintaining a flexible model of the state of the plant and the state of the student being trained. Steamer is very concerned with explanation as a primary goal. It needs to be able to represent the operating procedures for the steamplant in a form such that they can be executed and explained at different hierarchical levels. This execution and explanation should also be mimicked by the graphics.

Steamer illustrates state of the art explanation facilities. Similar explanation facilities will be included in many future systems. Much more work on multi-level systems will be done in the next decade.

4.0 SECOND GENERATION EXPERT SYSTEMS: AN APPROACH TO ELECTRONIC FAULT DIAGNOSIS:

Multiple or second generation expert systems are becoming more and more common, they tend to be highly structured and modular, incorporating knowledge from potentially diverse knowledge sources. Often these systems overcome the communications difficulties by means of a 'blackboard' [HAYES80] type mechanism, hence these systems are referred to as 'blackboard systems'.

The Hearsay-II Speech Understanding System [HAYES80] proposed a very interesting general framework for problem-solving in which a number of independent processes are coordinated to produce good cooperative behaviour. The general structure of Hearsay-II suggests an interesting possible approach to an advanced electronic fault diagnosis system. If we refer to figure 1 the effort in producing a multiple expert system, Hearsay-II, as opposed to a first generation expert system can be seen. The broken curve is a conjecture for future multiple level systems.

If the Hearsay-II approach to expert system structures were to be adapted for advanced general purpose electronic fault diagnosis systems, a potentially powerful expert system could be developed. For example, a hierarchical system for microprocessor based circuits, could consist of many clearly divisible sub-expert systems, at various levels of detail or abstraction. Examples of the possible sub-systems included might be:

a: Purely electrical checking at component level, driven by Kirchoffs laws, etc.
b: Digital testing at gate level, driven by test algorithms such as the D-algorithm [BENET82].
c: Electronic faults at IC level, eg propagation time delays due to characteristics of components or ICs.
d: Physical circuit construction faults, dry joints, etc.
e: Software analyses and checking.

A system of this type should be capable of extremely advanced fault and design error diagnosis, on a range of systems. As the logical representation of a digital circuit, is normally quite different and independent from the electrical representation of the same circuit, either logical tests/analysis or electrical tests/analysis on their own are not always sufficient for finding faults, particularly at the design stage. By creating a system which simultaneously examines the logical and electronic aspects of a particular circuit, the electronic

engineer's knowledge is being modelled much more accurately than in a system with a single level of abstraction or single fault finding/modelling approach. As a result a more "intelligent" expert system will be produced.

However the construction of a multilevel system as proposed here, involves coordinating diverse knowledge and data in a useful manner. This is extremely complex, even if it is limited to a very narrow problem domain. Such an expert system would overcome many of the limitations of plain rule based expert systems, and would allow for top down or bottom up approaches to solving the problem. As an expert system shell, or builder, it would allow for quicker and cheaper development of new expert systems, as the problem domain may be described or defined logically, electronically or partially at a series of levels, and interactions between the various levels or knowledge bases, would generate representations of the problem domain, or circuit, at each level of abstraction of the system. The various modules of the circuit, or fault domain, could be defined at the level, or in the terms which best suit them. The end result would be a multiplicity of representations, or models of the fault domain, within the expert system, each one representing different aspects or characteristics of the fault domain. The resulting system would obviously be very powerful, and could be produced relatively quickly within the expert system framework described here.

An example of such a system might be a fault or test system for microprocessor boards. Such a system could be quite complex and a variety of representations may be required. These representations of the system could include:

- o: The physical structure of the system.
- o: Electrical behaviour.
- o: Digital electronic behaviour.
- o: Digital logic representations.
- o: Representations of software running in the system.

The inter-relationship between the various representations might be illustrated with the following diagram:

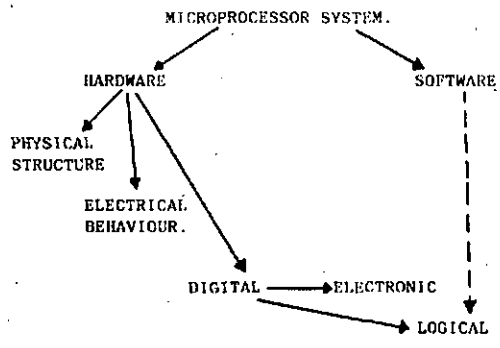


Fig 5.

Graphical representations of the components, and their behaviour along with graphical representations of the modes or states of the system would also be of importance in such a system.

5.0 ESPRIT PROJECT 96 - EXPERT SYSTEM BUILDER

The aim of Project 96 (a five year Esprit project) is to investigate the extent to which the production of Expert Systems can be industrialized. The partners in this project are CIMSA (France) with subcontractor LAIM, CSELT (Italy) with subcontractor Tecsiel, Soren T. Lyngso (Denmark) with subcontractor RISO, Plessey (UK) with subcontractors Plymouth Polytechnic and Trinity College Dublin (Ireland) [MNUCI84].

The project goes beyond the use of Skeletal Systems and uses a layered Conceptual Model to control complexity and separate concerns. An Expert System Builder (ESB) will be developed, and a number of prototype expert systems will be implemented. It is felt that expert system functions will be key competitive elements in most future electronic products. The underlying philosophical motivation for Project 96 comes from the claim that the main problem with expert system techniques becoming mass technology is less the basic theory than the sheer cost of implementation and the shortage of skilled manpower. Consequently the emphasis of the project is that the ability to plan and to fulfil plans has been given higher priority than the invention of new and more sophisticated representations of knowledge. It also wishes to eliminate the need for the knowledge engineer. The Layer Model of the ESB is given in Fig 6: [MNUCI84]

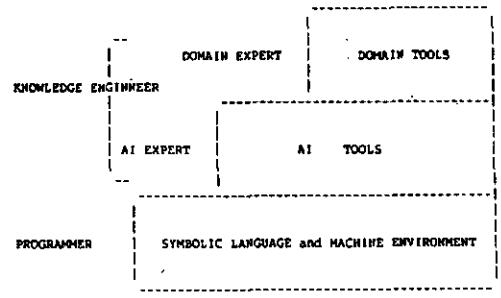


Fig 6: A Simplified Layer Model of the ESB.

A more detailed diagram is given in Fig 7: [MNUCI84]

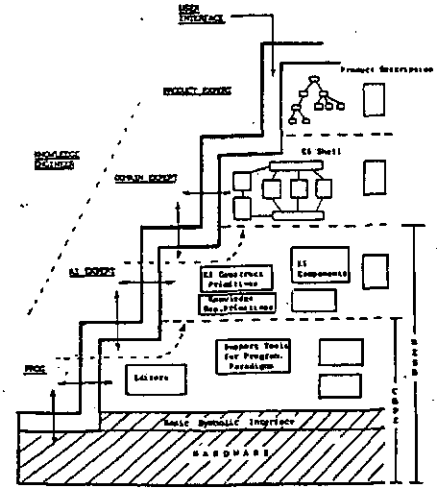


Fig 7: The Layer Model

A structure for the development of Knowledge Bases is shown in Fig 8 as a waterfall diagram. Of course, in practice the process is an iterative, not just a sequential, process with feedback to previous levels.

The layer model allows the ESB to incrementally improve itself by progressively building upon facilities in lower layers (so-called "technology bootstrapping"). The Common Base Programming Environment (CBPE) shown in Fig 7 is similar to the environment presently available on Lisp machines. The Basis Expert System Builder (BESB) includes both the symbolic and hardware environment, and the AI tools layer of Fig 8. At the top of Fig 7 there is an additional layer - product tools. The CBPE will provide primitive functions to support a set of high-level primitives e.g. primitives for creating and

editting frame structures will be needed toghether with a function for providing inheritance features.

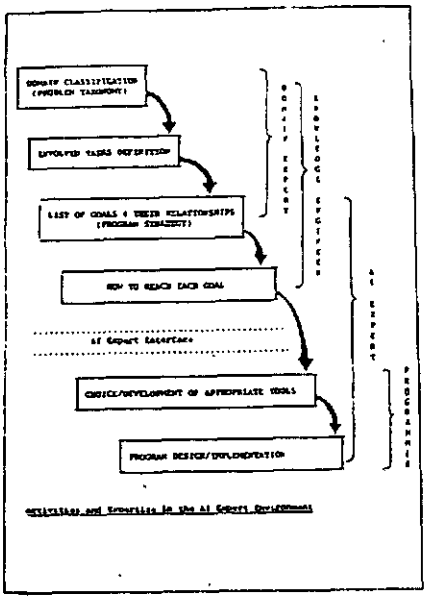


Fig 8: Activities and Expertise in building KBs

With respect to domain independence, the ESB can be considered as being made up of two blocks. These are the BESB (Basic ESB - which is domain independent) and the domain and product layers. This is shown in Fig 9 [MNUCI84] as the ESB layered model drawn in a ring fashion:

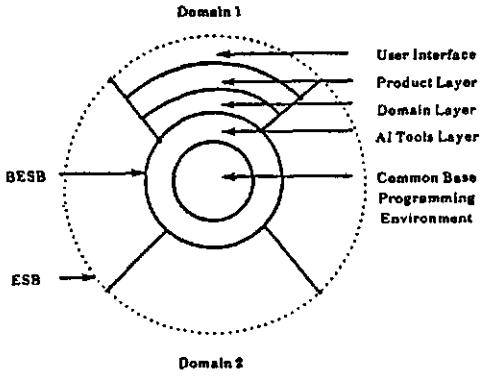


Fig 9: BESB domain independent Kernel of ESB

The three main areas of activity in developing the first version of the AI environment: Knowledge Representation and Manipulation; Search and Control primitives; and Knowledge system management tools.

KNOWLEDGE REPRESENTATION AND MANIPULATION:

The three basic knowledge representation formalisms chosen are:

-Logic with unification and resolution
-Object-oriented formalisms with inheritance and message-passing mechanisms.
-Production rules with pattern-matching and rule invocation.

These formalisms will be integrated in a fairly tight fashion in which each formalism may call the interpreter for the other.

SEARCH AND CONTROL PRIMITIVES:

BESB will contain a framework within the following can be flexibly specified:

- Definition of the search-space elements, the goals and initial states
- The state-changing operators and events
- Control strategy i.e. direction of search, conflict resolution, focus of attention, backtracking schema.

Metaknowledge or "knowledge about knowledge" is important especially for the control of search using problem-specific knowledge. A general concept is that every aspect of the search process should be accessible for reasoning at the meta-level.

KNOWLEDGE SYSTEM MANAGEMENT:

The BESB may be regarded as a collection of "Knowledge Systems" (KS's). A Knowledge System is a Knowledge Base plus a procedural component for manipulating the knowledge base. The BESB must contain a set of tools for the specification, creation, display and maintenace of KS's.

The KS approach is useful because:

- modularity allows difference sources with different formalisms and inference procedures
- Control or strategy knowledge may be separated from descriptive knowledge - Separates domain knowledge from product knowledge
- Techniques from other areas may be adapted e.g database concepts, "actor" formalism.

APPLICATION DOMAIN:

Four sample expert systems provide the framework and domain for experiment:

Soren T. Lyngso, RISO: an expert system for diagnosing an active power plant.
CIMSA, LAIM: an expert system to aid test pattern generation for complex digital boards.
Plessey, Plymouth, TCD: The diagnosis of an M68000 microprocessor system.
CSELT, Tecsiel: Diagnostic analysis of an MPC board for telephone equipment.

Each system makes use of both a deep, object-oriented knowledge base and a shallow, production-rule knowledge base. Each uses a hierarchy of abstraction levels to provide a flexible description of structure and function.

PRESENT STATUS:

A number of decisions and choices have been made:

- The BESB/ESB will be built using a commmercial AI tool
- The method to be used in developing the ESB is the "Topdown with Revision" method.
- The preferred hardware to support the CBPE is the Symbolics 3600.
- The preferred commercial AI tool chosen is ART, but further experiments and development will, in addition, use SIPROC developed by CIMSA.

PROGRESS:

Progress in the second year of the project could have been better, many of the problems encountered were contractual but these have now been overcome. This was due to specific project management policy of generating inter-personal contacts initially via meetings. Also very large use is made of Eurokom in the project. Communications have been good between all partners and subcontractors. Even though it is still early days in the project, the prospects of something viable being produced look very good. In the coming year it is expected that the four sample expert systems will be implemented. A number of interesting conclusions should be drawn from the simultaneous development of four expert systems.

6.0 GUIDELINES FOR A SUCCESSFUL EXPERT SYSTEM

This is a list of general guidelines for producing successful expert systems [BARS81] [MCDER83] [KUNZ84] [SMITH84].

- Focus on a narrow specialty area that does not involve common sense knowledge
- Task difficulty should be right i.e. not too easy (few minutes) or too hard (few hours) for a human expert.
- Commitment from an articulate expert
- Record a detailed protocol of the expert solving one prototypical case (watch the expert, don't just talk to him; also use the expert's terminology)
- Rapid prototyping
- Core set of representative problems
- Simple inference engine
- Avoid AI problem areas
- Document properly
- The process of building an expert system is inherently experimental
- Keep the problem-solution process visible
- Incremental development

The modest version of the system should have:

- Friendly interface: the user interface is CRUCIAL
- Gripe facility
- Library of cases
- Maintain the expert's interest

7.0 CONCLUSIONS

7.1 ELECTRONIC FAULT DIAGNOSIS SYSTEMS

The domain of electronic systems and electronic fault diagnosis appears to be particularly suitable for applying expert systems to. This is due to a number of important points.

- Standard and nature of test equipment: The test equipment used in this domain generally gives immediate results, which are accurate quantitative values. This differs from the situation in medical diagnosis, where results are not always immediate and are often qualitative rather than quantitative. Accurate results have the effect of reducing uncertainty, and the resulting expert systems should rely less on certainty factors.

Certainty factors were not really needed in the development of the Processor system, this is also due to the preciseness of the test equipment and the definite results of tests carried out, and is an indication of the suitability of the electronic fault domain to the development of expert systems.

- The Predictability of Circuit Behaviour: Electronic systems tend to be predictable, in so far as given a set of inputs, the outputs from a working electronic system can be theoretically predicted with great accuracy. Furthermore, the effect of a particular fault or set of faults on the behaviour of an electronic system can also be predicted with a high degree of accuracy in many cases. This is not the case in Medicine which has attracted a great deal of Expert Systems research, where the effect of a given drug on a particular patient can be speculated about but not accurately predicted.

- Simulation of Electrical Systems:

Electrical Systems can be simulated with a high degree of confidence. Where the characteristics and behaviour of the components and subsystems and the interaction between them can be examined in expanded time or real time. This aspect of electronic circuits is related to their predictability but distinct from it, and contributes to rule production, rule validation and eventually to expert system validation.

These aspects of electronic circuits allow good models to be constructed, which can be used to support the knowledge base, if not form an intrinsic part of it. They allow the representations of the system in the knowledge base to be substantiated, and are useful for developing or expanding knowledge bases.

7.2 CONSTRUCTING EXPERT SYSTEMS

Throughout the construction of the Processor expert system, it became apparent that the main problem in developing an expert system was development of the knowledge base. This was clearly a two stage process, firstly collecting the data or knowledge, and secondly encapsulating this usefully in rule form. Normally the first task would be carried out by the domain expert, and the second by the knowledge engineer. Among the problems which the domain expert faces, are ensuring the completeness of the knowledge and its accuracy or validity. The knowledge engineer must ensure that the representation of the knowledge in the computer is what the domain expert actually intended, in other words that it is a valid representation of the knowledge. Along with being valid, the knowledge must be structured in a useful manner. Rules for instance must be well designed and structured in order to be of use in a consultation.

Perhaps the largest single difficulty in developing an expert system is overcoming communications difficulties between the domain expert and the knowledge engineer. Each of these may understand little about the other's field, and misunderstandings are inevitable. There is no apparent way of overcoming this problem, and it could be a serious limiting factor in the rate of development of new expert systems. Some proposed systems will allow the domain expert to input knowledge directly to the expert system builder, but the problems of completeness and validity still exist, along with the problems the domain expert will have communicating with the computer. Such an approach automates the knowledge engineering process, but does not overcome the communications problems.

It is said that the only limiting factor in the development of new expert systems, is the problem of gathering domain knowledge or expertise on which to base those systems.

7.3 Future Directions

Expert systems have been around for quite some time e.g. Mycin was built in 1974, but in the last few years have found their first applications outside artificial intelligence research laboratories. This transition from research laboratories to real world application is being helped by several factors:

- Artificial intelligence workstations (nearly all single user) are becoming available (see Table 3) although they are still very expensive. VLSI developments have accelerated the development of these machines.
- Lisp has become relatively standardized, much better documented, and available on mainframes and also on personal computers such as the IBM-PC and PC-compatibles. Common Lisp is an attempt to standardize. Prolog has also become available in similar fashion. These languages are available at a modest price ($500 - $1,000) for personal computer applications.
- Expert System Shells for implementing expert computer systems are rapidly becoming available in a price range from $100 to $100,000 (see Table 1 for a selection). In general, the facilities and quality of the shell is proportional to the cost, and the well-known legal adage applies - CAVEAT EMPTOR.

Some problem areas remain:

- The cost and amount of work needed to build an expert system is possibly up to $1m for a 10 to 25 manyear system. To build even a small production rule system to commercial standards, has a lower bound of 5 manyears.

with at least 2 people involved. The expert shells now available will go some way towards solving this problem. We have also seen from Fig 1 that a fairly dramatic learning affect seems to apply.

- Shortage of skilled manpower as exemplefied in Fig. 2, as well as shortage of knowledge engineers. There is also a shortage of AI programmers.
- A second major problem is the amount of time necessary to acquire knowledge from an expert in some problem domain. This problem is due to a lack of tools for the task, but is also due to our own inadequate understanding of human problem-solving.
- Finally there is considerable experience in industry of the problems of developing and maintaining large software systems, even if there is not a lot of satisfaction with the resulting software quality. However there is little experience as yet of the long-term maintenance of expert systems the R1 case study discussed previousely is interesting for this purpose. There is also little experience in handling the evolution of such systems, while still keeping them user-friendly. Some lessons from software engineering may be appropriate.

There is no doubt that expert systems of different shapes, sizes, and applications will have a major impact on industry and services in the next five to ten years. Possibly the major impact of the microcomputer (r)evolution will be through the use of expert systems.

Table 2: Some Expert Systems Developed at TCD.

NAME OF SYSTEM	LANGUAGE OR BUILDER	NUMBER OF RULES	CURRENT STATUS AND COMMENTS.
LENDING	EMYCIN	18	Developed by Mark Lysaght, the system evaluates applications for loans and offers advice to the bank manager. Lending is currently used as a demonstration system.
PROCESSOR	EMYCIN	213	Developed by John Gleeson. Processor carries out electronic fault diagnosis on a particular range of microprocessor boards and is currently available to students developing these boards as part of their course work.
FLOORS	EMYCIN	50	Floors is an experimental system developed by Niamh Harty, to aid structural design of buildings. Current work on more advanced versions of this system is being done in LISP and not Emycin.
KRCPI	Franzlisp	1200 (but rules have a very simple format)	Developed by Padraic Broanna, this system does thyroid malfunction diagnosis A more developed version of this system will be ported from a VAX to a Rainbow with Commn Lisp in St.James's Hospital Dublin, where it will be used.
CarFaults	PROLOG	150	Developed under Mike Brady. An objective was to evaluate Prolog for expert system development.

Table 2: Some Expert Systems Developed at TCD.

APPENDIX. Tables 1 to 3.

Table 1. Some Expert System Shells Available in Mid-85.

Name	Cost	Number Sold*	Language	Hardware	X.E.	Applications
APES	$250	400	Prolog	IBM-PC	Yes	Medicine, Legal,Social Security, Civil Engineering
ART	$85,000	40	Lisp	Symbolics, Lisp M/c, Vax	Yes	NASA space shuttle landing system, fault diagnosis in missiles.
Envisage	$12,000	5	Pascal	-	Yes	
ESP Advisor	$600	100	Prolog	-	Yes	
Expert Ease	$700	600	Pascal	IBM-PC	No	
Extran-7	$18,000	6	Fortran	IBM PC	Maybe	Circuit Board Fault Finder; Seismic Data Analysis Front End Oil & Aerospace Applics
ICL Advisor	$11,000	8	Pascal	ICL	No	Computer sizing; Fault Diagnosis in telephone and manufacturing.
ICL Reveal	$30,000	7	Fortran	main-frame	No	Capacity planning; Energy planning; Human response mgmt.
KEE	$50,000	30		Symbolics Lisp M/c Xerox	Yes	Being used by Boeing and Logica.
Personal Consultant Reveal	$2,000	400	Basic Fortran	TI Prof IBM, Dec, PC, Prime		Market modelling; Vax Performance monitoring; Corporate analysis.
Rulemaster	$20,000	2	C, Unix	Vax,Sun	Yes	Severe storm forecasting; Power transformer diagnosis; NASA
Savoir	$10,000	5,000	Pascal	Vax	Yes	Individual use fungicide; Stress/cracking/advice system; Production control
	$3,000			IBM-PC		
TIMM				IBM PC	No	
R1	$495				No	

Table 1. Some Expert System Shells Available in Mid 85.

* No. Sold is an estimate as of mid-85.
Note that some in some cases copies have been sold to parent/shareholder companies.

IBM PC means an IBM PC or PC compatible machine. It may also include other machines such as Sirius, Dec Rainbow, Apricot etc.

Table 3 Some AI Workstations Available Mid-85.

Name	Cost	Main Language	Other Languages	Source	Comments
Advent AI	$40,000	Common Lisp	Pascal,C	Advent Data Systems	
Explorer				Texas Instruments	
XPS iD	$100,000-$300,000	Zetalisp	Unix 4.3	Recal Norsk	
Lambda	$100,000 $200,000	Zetalisp		Lisp Machines Incorporated	Up to 4 users
Symbolics 3600	$70,000 $125,000	Zetalisp Flavors	Prolog, Lisp(s), Fortran, Pascal, Nacayma, ANT, KEE	Symbolics Inc.	Prolog & Lisp can call each other
Tektronix 4404	$17,800	Smalltalk 80	Franzlisp, Prolog	Tektronix	
Xerox 1108	$13,000 $40,000	Interlisp D	LOOPS	Xerox	Lisp Object Oriented Programming System

Table 3 Some AI Workstations Available Mid-85.

N.B. Number sold is an estimate, as of mid-85.
Note that some sold may be to parent/shareholder company.

REFERENCES:

[BARS81] D.R.Barstow and B.G.Buchanan,
"Maxims For Knowledge Engineering",
HPP-81-4, 1981.

[BENET82] R.G.Bennetts
"Introduction To Digital Board Testing"
Arnold 1982.

[BRADY84] M.Brady,P.Agre,D.Braunegg and J.Connell,
"The Mechanics Mate"
ECAI 84 (ed: Tim O'Shea)
North Holland, 1984.
(pp 681-696).

[CLARK82] Clark and Tarnlund,
"Logic Programming"
Academic Press, 1982.

[COX84] I.J.Cox,
"Expert Systems"
IEE "Electronics and Power".
March 1984. (pp 237-240).

[FEIGE79] E.A.Feigenbaum,
"Themes And Case Studies Of Knowledge Engineering",
in "Expert Systems In The Microelectronic Age",
ed D.Michie,
Edinburgh University Press, 1979, (pp 3-25).

[FEIGE81] Barr and Feigenbaum,
"The Handbook Of Artificial Intelligence"
Pitman Books, 1981.

[FEIGE83a] E.Feigenbaum and P.McCorduck,
"The Fifth Generation - Artificial Intelligence And
Japan's Computer Challange To The World"
Addison Wesley, 1983.

[FEIGE83b] E.Feigenbaum,
"Knowledge Engineering: The Applied Side"
in "Intelligent Systems, The Unprecedented Opportunity"
Ellis Horwood, 1983.

[FREN85] K.A.Frenkel,
"Twords Automating The Software-Development Cycle",
COMM ACM, June 1985, (pp.578-589).

[HAYES80] Erman,Hayes-Roth,Lesser and Reddy,
"The Hearsay-II Speech Understanding System:
Integrating Knowledge To Resolve Uncertainty"
Computing Surveys (ACM), Vol 12, June 1980.
(pp 213-253)

[HAYES83] edt. Hayes-Roth,Waterman and Lenat,
"Building Expert Systems"
Addison Wesley, 1983.

[IEEEC83]
"Special Issue On Knowledge Representation".
IEEE Computer, October 1983.

[JOHN84] T.Johnson,
"The Commercial Application Of Expert System Technology"
OVUM, London, 1894.

[KUNZ84] J.C.Kunz, T.P.Kehler and M.D.Williams,
"Applications Development Using A Hybrid AI
Development System"
AI Magazine, Fall 1984, (pp 41-54).

[MNUC184] F. Manucci, J. Gallagher, M. Swabey
and C. Fisher
"Expert System Builder"
Esprit '84: Status Report of Ongoing Work
ed. J. Roukens and J. F. Renuart
Elsevier Science Publishers
1985

[MCDER82] J.McDermott,
"R1:A Rule Based Configurer Of Computer Systems",
CMU-CS-80-119, April 1980.

[MCDER83] J.McDermott,
"Extracting Knowledge From Expert Systems"
IJCAI 1983, (pp 100-107).

[MCDER84] J.McDermott and J.Bachant,
"R1 Revisited: Four Years In The Trenches"
AI Magazine, Fall 1984, (pp 21-32).

[MICHE79] edt. D.Michie,
"Expert Systems In The Microelectronic Age"
Edinburgh University Press, 1979.

[MILSN85] R H Michaelsen, D Michie, A Boulanger,
"The Technology Of Expert Systems"
Byte April 1985 pp303-312.

[NAU83] D.S.Nau,
"Expert Computer Systems"
IEEE Computer, Febuary 1983, (pp 63-85).

[NILSN82] N.Nilson,
"Principles Of Artificial Intelligence"
Springer-Verlag, 1982.

[OSHEA84] T.O'Shea,
"Artificial Intelligence"
Harper And Row, 1984.

[SHORT75] Shortliffe,
"Computers And Biomedical Research"
Elsevier 1975.

[SHORT76] E.H.Shortliffe,
"Computer-Based Medical Consultations:MYCIN"
Elsevier Computer Science Library, 1976.

[SMITH84] R.G.Smith,
"On The Development Of Commercial Expert Systems"
AI Magazine, Fall 1984, (pp 62-73).

[STEFK83] M.J.Stefik and J.De Kleer,
"Prospects For Expert Systems In CAD"
Computer Design, 22, 1983, (pp 56-76).

[STONE84] T.Moto-oka and H.Stone,
"Fifth Generation Computer Systems: A Japanese Project"
IEEE Computer, Vol 17 Number 3, March 1984. (pp 6-13).

[SWABY85] ed. by M.Swabey
"Expert System Builder - The Second Year"
In Esprit '85: Status Report of Ongoing Work
to be published in 1986 based on Esprit Technical week
September 1985.

[TOURE84] D.Touretzky,
"LISP, A Gentle Introduction To Symbolic Computation"
Harper and Row 1984.

[VANML81] Van Melle,Scott,Bennett and Peairs,
"The EMYCIN Manual"
Dept. Of Computer Science, Stanford University.
Report Number STAN-CS-81-885, October 1981.

[WINST79] H.Winston,
"Artificial Intelligence"
Addison Wesley, 1979.

TEHNOLOGIJA, ARHITEKTURA IN ZGRADBA RAČUNALNIŠKIH SISTEMOV

TECHNOLOGY AND SYSTEM ARCHITECTURE

RELAXATION MESH DYNAMICS IN THE METHOD OF FINITE DIFFERENCES

Alojz Paulin, Niko Bezić
Department of Technology, University of Maribor
Bojan Jenko
IEVT, Ljubljana

UDK: 519.852

The number of relaxation mesh points in the finite difference method is defined and essentially restrained with the fast memory size of the computer in use. The boundary conditions for the second order differential equation of the first degree include in many technical applications certain fine structure in their geometry. With an uniform relaxation mesh a given finite number of mesh points that fine structure can not be taken properly into consideration. The relaxation mesh dynamics represents a procedure which gives some possibilities for the improvements. The applicability and the constraints of the relaxation mesh dynamics are quoted. The uniqueness of the solution to the second order differential equation is mentioned with regard to the application of the relaxation mesh dynamics.

The method of finite differences in its computational version yields a digital solution to the second order differential equation of the first degree. It provides in a certain way the uniqueness of that solution. The solution uniqueness could be achieved with an algorithm, which should exactly define all the steps in the numerical approach to the solution of the differential equation.

The finite fast memory size in the computer defines and restrains the number of mesh points in the relaxation mesh.

There is a possibility to increase the number of mesh points in a given small part of the region, and simultaneously decrease the number of mesh points in the rest of the whole region given within the extensions of the independent variables. Those extensions are defined with the boundary conditions to the differential equation.

The uniqueness of the solution cancels, when the mesh thinning and/or the mesh thickening is performed optionally and independently from the form of the differential equation solution. A complete uniqueness of the solution should be achieved, and including the relaxation mesh dynamics, when the mesh dynamics could follow the equipotential curves of the differential equation expected solution.

The basic suggestion how to approach the relaxation mesh to the equipotential curves of the expected solution for a given differential equation and the boundary conditions is described by A.M. WINSLOW in the year 1966. A successful application of such a relaxation mesh dynamics is usually connected to the interactive graphics at the computer in use. That application is introduced by J.S. COLONIAS in the year 1967.

When designing the relaxation mesh dynamics in the procedure, where the boundary conditions should be satisfied, the logical lay-out has to be strictly distinguished from the lay-out of the real geometry, given with the boundary conditions of the problem under consideration. With another words, one has to distinguish the boundary conditions input data as given into the programme from the data about the boundary conditions, accepted within the programme performance. The relaxation mesh dynamics has to be adjusted to the lay-out of the real geometry of the problem under consideration.

Therefore the relaxation mesh generator should provide the solution uniqueness in the differential equation as well as the mesh dynamics.

It should be done in such a way, that the solution accuracy is evenly distributed over the whole region, given within the extensions of the independent variables and defined with the boundary conditions.

In this report, we do intend to describe some experiences with the relaxation mesh dynamics in the method of finite differences, in the computational work, done during the past several years.

The simplest approach of the relaxation mesh dynamics represents the partitioning and the compounding of the

solution matrix, respectively. Usually the matrix partitioning and the matrix compounding are performed simultaneously, although not necessarily, with the thinning and/or the thickening of the relaxation mesh, respectively. That approach includes the solution value linear interpolation at the transition from the thicker mesh to the thinner one. The matrix partitioning and/or compounding involves very large data handling.

The next approach in the relaxation mesh dynamics is the programmed mesh thinning. The interpolation conditions require, the region of the thinner mesh has to be surrounded and located inside the preceding thicker one. In this report some of that approach significances with the programmed mesh thinning are described.

It is possible to programme as many mesh thinning as they could be accepted in the finite fast memory size of the computer central processor. At the RRC CYBER 172 computer the procedure with five sequential mesh thinning is implemented. The basic mesh extension could be that way decreased 2^4 times. That appears as a satisfactory solution to the most of the problems under consideration.

At each of the mesh thinning, it is firstly necessary to determine the interpolated solution values of the differential equation at the thinner mesh region boundary. With the defined solution values at the region boundary for each of the relaxation mesh thinning, an interation procedure for a given kind of relaxation is performed. The kind of relaxation, i.e. the over-relaxation and/or the under-relaxation depends on the linearity and the non-linearity of the problem, respectively. The actual matrix of the differential equation solutions for a given relaxation mesh thinning is saved on the magnetic tape for the later final consideration.

It is necessary to remark, the mesh thinning could be simply developed only for the method of finite differences at the accuracy of second order. At the accuracy of the fourth order, as introduced by P. BONJOUR and S. NATALIS in the year 1972, the relaxation mesh thinning would require an extremely involved procedure.

The linear interpolation of the values for the differential equation solution at the mesh thinning does not appear always as a justified one. An interpolation with higher order splines should yield a more applicable results.

The relaxation mesh dynamics provides a better interpretation of the boundary condition fine structure and a more uniform distribution of the solution accuracy over the whole region under consideration. A better accuracy of the unique solution in the whole region could be accomplished with the method of finite differences of the fourth order and applying an exact consideration of accurately defined boundary conditions.

The suitable and an applicable method of finite differences using the accuracy of the second order should include several ten thousand of mesh points in the relaxation mesh. The solution accuracy is directly proportional to the inverse value of the mesh points number in the relaxation mesh.

For the purpose of completeness and briefness this report is supported with only two references. The report [1] from the Dubna Institute is quoted for the early publications and historical background. The work by P. BONJOUR [2] includes more recent achievements, particularly the intentions how to approach the relaxation mesh dynamics to the form of the expected solution of the second order differential equation.

REFERENCES

[1] S.B. VOROZHTSOV et al.
"Calculation of a two dimensional magnetic field using irregular triangular mesh", Report P9-5013, Joint Institute for Nuclear Research, Dubna (1970)

[2] P. BONJOUR
"Computing electrostatic and magnetic fields", pp. 1-44 in the Advances in Electronics and Electron Physics, Supplement 13A, Applied Charged Particle Optics, A. Septier, Editor, Academic Press (1980) New York

VME VODILO V VEČRAČUNALNIŠKIH ARHITEKTURAH

M. COLNARIČ, I. ROZMAN, B. PREMZEL
TEHNIŠKA FAKULTETA MARIBOR

POVZETEK - Članek obravnava problematiko vodil v večprocesorskih sistemih s poudarkom na VME vodilu. To je prirejeno za prioritetni in Round Robin arbitracijski algoritem z Daisy-Chain povezavo na posameznih nivojih. Ker je v novejših raziskavah utemeljeno, da tovrstni algoritem ne daje optimalnih rezultatov, so v članku nakazani možni načini realizacije drugih izbirnih pravil.

ABSTRACT - In the article multiprocessor systems' buses are discussed with accent on the VME bus. It is designed for priority and Round Robin arbitration scheme with Daisy - Chained modules on each level. Since it is proved in recent research that this kind of algorithms are not optimal, some other arbitration protocols are described.

UVOD

Z razvojem mikroprocesorjev prevzemajo mikroračunalniški sistemi čedalje kompleksnejše naloge. Marsikaterih zahtev pa tudi najmočnejši procesorji ne morejo zadovoljiti. Iz tega razloga in ker jim cena nenehno upada so čedalje zanimivejši sistemi, kjer si več procesorjev deli globalne resurse.

Najpomembnejši med njimi je vodilo za prenos podatkov. Preko njega procesorji komunicirajo z drugimi resursi in med seboj. Vsak sistem z več procesorji mora reševati problem dodeljevanja vodila uporabnikom. Uspešnost reševanja tega problema v veliki meri vpliva na prepustnost celotnega sistema in je torej ena od najvažnejših odločitev pri snovanju mikroračunalniškega sistema. Hitrost je tudi vzrok, da ne uporabimo programske temveč aparaturno rešitev. Sistem za dodeljevanje vodila po vnaprej določeni shemi se imenuje arbitracija.

Druge pomembne odločitve pri snovanju protokola vodila so še

- sinhrono ali asinhrono vodilo: odločiti se moramo, ali bodo vse enote v računalniku delale z istim taktom, ki se prenaša po skupnem vodilu (sinhrono) ali bodo lahko njihove hitrosti različne (asinhrono). Sinhroni način je bistveno enostavnejši, asinhroni pa omogoča kasneje vključevanje novejših, hitrejših enot.
- multipleksirano vodilo ali ne: ker zaradi same zasnove mikroprocesor in resursi ne potrebujejo istočasno na linijah naslova in podatkov, lahko po istih linijah pošljemo naprej adreso, nato pa vsebino teh celic. Zaradi manj linij je vodilo manjše, uporabi se manj vezij, vendar ne dosegamo takšnih hitrosti kot pri nemultipleksiranih vodilih.

Članek teh problemov ne obravnava. Dotaknili se bomo le tematike arbitraže.

SPLOŠNO O ARBITRACIJSKIH ALGORITMIH

Glavni nalogi arbitracijskega sistema sta preprečiti istočasno dostop dveh ali več procesorjev do vodila in dodeljevanje vodila procesorjem po vnaprej predpisanem optimalnem pravilu.

Osnovne odločitve o tipu arbitracijskega algoritma so vezane na naravo naloge, ki bi jo naj opravljal. Če gre za sprotni sistem, nas bo zanimal prioritetni (ali vsaj delno prioritetni) algoritem, če želimo poslovni sistem z več procesorji in pametnim razdeljevanjem nalog, bo nam bolj ustrezal FIFO ali Round Robin algoritem (rotirajoči prioritetni algoritem).

Kot za pridobitev nas zanimajo tudi algoritmi za sproščanje vodila. Poznamo dva glavna principa

- sprostitev, ko je prenos končan (release when done, RWD) in
- sprostitev na zahtevo (release on request - ROR).

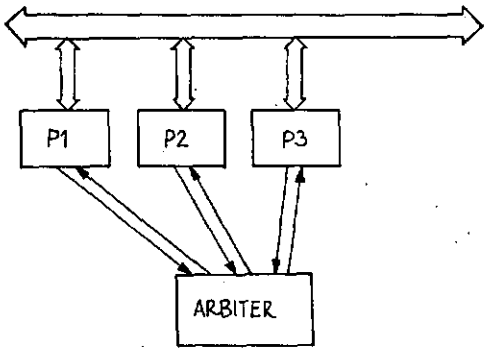
Pri prvem enota, ki je zmagala v arbitracijskem postopku, sprosti vodilo, ko je izvedla prenos podatkov, ne glede na to, katera enota je medtem zahtevala vodilo. Pri drugem pa enota med prenosom podatkov spremlja linijo, preko katere ji arbiter signalizira, da mora prekiniti s

komunikacijo, ker je vodilo zahtevala po prioriteti višja enota.

Iz samega delovanja vodila lahko sklepamo, da poleg vodila in čistih modulov obstaja še sistem, ki skrbi za arbitracijo. Ta arbitracijski sistem je lahko samostojni modul, lahko pa je porazdeljen na vse potencialne uporabnike vodila. Glede na način povezave modulov nanj ločimo nekaj osnovnih tipov:

- Zvezdasta povezava:

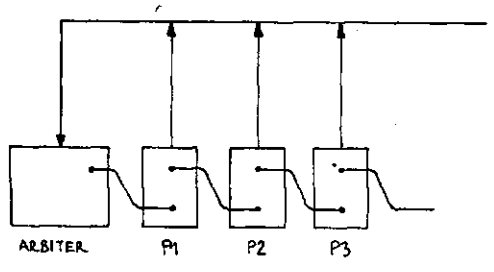
  Vse naprave, ki lahko zahtevajo vodilo, so povezane na arbiter mimo vodila s po dvema posebnima linijama. Po eni enota zahteva vodilo (Bus Request), po drugi dobi odobritev (Bus Grant). Protokol je izredno enostaven, arbiter lahko dela po kakrčnikoli metodi. Slabe strani so drago posebno ožičenje, na vodilu ni informacije o tem, kdo trenutno zaseda, na arbiter ni mogoče programsko vplivati.



Slika 1: Zvezdasta povezava na arbiter

- Daisy chain:

  Na vsakem konektorju je par priključkov za arbitražo. Signal s prejšnjega konektorja gre na en priključek, z drugega pa na naslednji konektor. Na ta način je realizirana zaporedna povezava modulov.



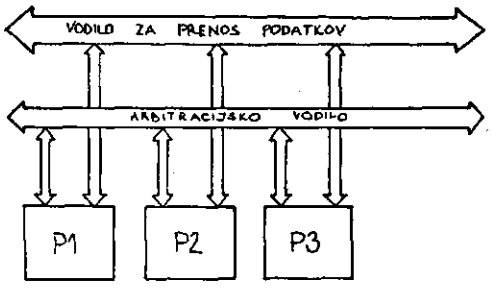
Slika 2: Daisy chain

Kadar hoče naprava vodilo, pošlje po "wire - OR" liniji signal arbitru in čaka na odobritev na vhodnem priključku. Vsak modul ima vhodni in izhodni priključek kratko staknjen, razen če tudi sam želi vodilo. Kadar torej dve ali več naprav istočasno pošljeta zahtevo po vodilu, ga bo dobila tista, katere konektor je fizično bližje arbitru.

Slabosti tega sistema so:
- počasnost, saj vsaka plošča doda nekaj zakasnitve, dokler se signal ne vrne do enote, ki je zahtevala vodilo
- v vsakem konektorju mora biti plošča ali slepi konektor s prevezo, ki omogoči prenos signala
- o arbitraciji je na liniji še vedno malo podatkov.

- Shema z naslovno - prioritetnimi linijami

  To posebno shemo uporablja več sodobnejših vodil in tudi standard IEEE P 896 Futurebus. Na vodilu je 4 do 8 arbitracijskih linij, na katere skušajo posamezni moduli vpisovati svoje prioritetne številke po posebnem algoritmu, opisanem v 4. poglavju. Čigar številka je ostala na vodilu, je dobil odobritev za prenos podatkov. Razširitev osnovnega algoritma, da se moduli po principu "poštenosti" sami odpovedujejo vodilu in s tem omogočajo, da tudi moduli z nižjimi prioritetnimi številkami dobijo dostop do vodila, bistveno izboljšanje lastnosti, povezane s sodobnimi izsledki o primernosti posameznih arbitracijskih algoritmov. S tem se namreč približujemo enakopravnosti modulov. Prioritete ostanejo le za reševanje istočasnih zahtev vodila.



Slika 3: Naslovno prioritetna arbitraža

ARBITRACIJSKI ALGORITEM PRI VME VODILU

Eno najbolj znanih vodil je gotovo VME vodilo. Njegov razvoj sega v pozna sedemdeseta leta, ko je pri Motoroli skupina inženirjev razvijala Exormacs na osnovi

MC 68000 procesorja. Njegovo vodilo se je imenovalo VERSAbus. Kmalu pa so v evropskih podružnicah Motorole ugotovili, da se VERSAbus ne bo obnesel kot standard, zato so ga prenesli na evropski format tiskanega vezja in ga imenovali VME. Leta 1983 je IEEE začela postopek za standardizacijo VME vodila ter mu dala šifro P 1014. Sprejetje standarda se pričakuje konec leta 1985. Istočasno bi ga naj sprejela tudi IEC.

VME sistem je sestavljen iz štirih skupin signalnih linij (vodil) in zbirke funkcionalnih modulov, ki predstavljajo vmesnike med napravami in vodili.

Štiri vodila, ki sestavljajo VME vodilo, so vodilo za prenos podatkov (DTB - Data Transfer Bus), arbitracijsko vodilo, prioritetno prekinitveno ter vodilo za notranje potrebe (ura, inicializacija, napake ipd...).

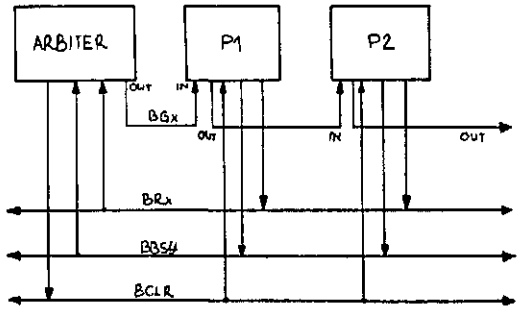
Vodilo je bilo prvotno namenjeno enoprocesorskim sistemom, v katerih so določene enote lahko zahtevale prenos podatkov (npr.DMA kontrolerji, disk kontrolerji itd...). Z razvojem večprocesorskih sistemov pa se VME vodilo uporablja tudi zanje.

Arbiter na VME vodilu je lahko izveden kot ena od treh opcij:

- PRI (prioritetni algoritem): arbiter dodeli vodilo po stalnih prioritetah modulov, ki dajejo zahteve na linije, od najvišje (BR3) do najnižje (BRO)
- RRS (Round Robin Select): arbiter dodeljuje vodilo po principu rotirajočih prioritet. Če je imel vodilo modul na nivoju N, bo imel najvišjo prioriteto modul na nivoju N-1, dokler ne pridemo do začetka (N = 0). Tedaj ima najvišjo prioriteto spet N = 3.
- ONE (edini nivo): arbiter upošteva samo linijo BR 3 in obravnava Daisy Chain strukturo na tem nivoju.

Arbiter je vedno v fizično prvem konektorju na vodilu. To je povezano s principom Daisy Chain. Lahko je samostojna plošča ali pa le razširitev univerzalne procesorske plošče, ki izpolnjuje zahteve minimalnega arbitra.

Razen pri opciji ONE arbiter sprejema zahteve po vodilu na štirih linijah BRØ do BR3, ki so izvedene v tehniki z odprtim kolektorjem, tako, da lahko povežemo izhode več modulov. Vsaka linija ima odgovarjajočo linijo za odobritev vodila (BGx-grant), izvedeno kot Daisy Chain. Če je vodilo prosto, ko je arbiter dobil zahtevo, takoj pošlje grant po ustrezni liniji. Signal potuje med moduli, ki imajo vhodne in izhodne linije sklenjene, dokler ne naleti na modul, ki je nanj čakal. Ta dobi vodilo. Ko ga je nehal uporabljati, ga sprosti in pošlje grant za naslednjo zahtevo. Informacijo o zasedenosti vodila nosi linija BBSY (Bus Busy). Arbiter lahko pri prioritetnem načinu tudi zahteva takojšnjo sprostitev vodila po liniji BCLR (Bus Clear), kar odgovarja načinu sproščanja na zahtevo (ROR).



Slika 4: Shematski prikaz delovanja arbitracijske sheme na VME vodilu za eno Daisy Chain linijo

Za arbitražo lahko uporabimo prioritetni ali krožni (Round Robin) algoritem. Na vsakem od nivojev se ustvari Daisy Chain struktura, pri čemer ima modul, ki je bližje arbitru, višjo prioriteto, oziroma bo večkrat dobil odobritev za uporabo vodila. Novejše raziskave kažejo, da prioritetni način ni najoptimalnejši pri dodeljevanju resursov. To ne velja za sprotne sisteme, kjer zaradi narave taskov mora obstajati prioritetna lestvica. Največja prepustnost sistemov se doseže pri enakovrednosti procesorjev in ustrezni razdelitvi nalog med njimi. V ta namen bi bil najustreznejši arbitracijski algoritem čakalna vrsta po principu FIFO.Te pa s konceptom, kot ga ima VME vodilo, ni mogoče uresničiti.

Sam problem nastopi pri tesni povezanosti procesorja v sistemu. Če dodamo vsakemu procesorju svoj del lokalnega pomnilnika in tako samostojne naloge, da ne bo pogosto zahteval povezave z drugimi moduli, bo vodilo razmeroma nezadostno in se bo vrsta le redko zgradila. Tedaj strežni algoritem ne bo imel prehudega vpliva na prepustnost sistema. Tipični primer takšne arhitekture so računalniški sistemi za vodenje robotov.

Pogosta je zasnova, kjer vsako os robota vodi svoj računalnik, vsi pa so povezani preko VME vodila s skupnimi globalnimi resursi - globalni pomnilnik, disk in druga periferija. Vsak računalnik je na nivoju premikov svoje osi avtonomen, ima svoj lokalni pomnilnik in direktno upravlja preko vmesnikov z izvršilnimi organi. Skoplje-

nost sistema je v tem primeru še dovolj lahka, da slabe strani prioritetne sheme ne pridejo usodno do izraza.

SPLOŠNEJŠI ARBITRACIJSKI ALGORITEM

Ker so večprocesorski sistemi vse številnejši, je nastala potreba po novem vodilu, namenjenem prav zanje. V ta namen so pri IEEE ustanovili posebno komisijo, ki razvija novo standardno vodilo IEEE P896.

Osnovna ideja o arbitraciji datira v leto 1966. Kasneje je bila predelana in prirejena za sodobne zahteve ter uporabljena v standardih IEEE 696, P896 ter v NuBus (Texas Instruments) in Multibus II (Intel).

Vsaka enota ima svojo N-bitno arbitracijsko številko, vodilo pa N-bitno arbitracijsko vodilo na principu odprtega kolektorja. Ko dobi enota, ki je zahtevala vodila, signal za začetek arbitracije, skuša na arbitracijske linije vpisati svojo številko. Če je na vodilu že višja prioriteta, kot je njena, umakne manj pomembne bite. Enoti, katere prioriteta ostane na vodilu ob koncu arbitracije, se dovoli: prenos podatkov. Predno ta sprosti vodilo, sproži ponovno arbitracijo, ki omogoči naslednji najpomembnejši enoti dostop do vodila.

Ker smo s tem pravilom še vedno pri prioritetah mu dodamo še pravilo "poštenja", ki prepoveduje enoti, ki nima izrecne časovne omejitve, zahtevati vodilo, ki ga je sprostila, dokler ga ne dobijo vsi drugi prosilci. Na ta način dobimo gotovo najpopolnejšo vodilo, vendar z najbolj zapletenim arbitracijskim vezjem, ki mora biti na vsakem modulu v sistemu.

Na tem mestu je treba razčistiti s pojmom prioritete na vodilu. Namen prioritet v zvezi s pravilom "poštenosti" je rešiti problem istočasnih zahtev. V zgoraj opisanem sistemu ima prioriteta malo skupnega z odločanjem, katera enota bo največkrat dobila vodilo in nima nič skupnega s prioriteto taskov. Taski z nizko prioriteto imajo mnogokrat na vodilu visoko prioriteto (masovni prenosi podatkov itd...).

ZAKLJUČEK:

VME vodilo je bilo koncipirano za enoprocesorske sisteme. Sam koncept sicer podpira tudi večprocesorsko arhitekturo, vendar ne moremo doseči optimalne propustnosti, kot jo teoretično lahko določimo. VME sistem pa ima druge prednosti. Ker je kot standard izredno široko razširjen in priznan, obstaja na tržišču mnogo modulov, ki jih lahko uporabimo. Od zahtevnosti problema, ki bi ga radi rešili z našim računalnikom bo odvisno, ali se bomo zadovoljili z VME ali bomo iskali optimalnejše arbitracijske algoritme.

LITERATURA

/1/ Rozman, I., Colnarič, M.: Modeliranje MP/MC arhitektur s skupnim vodilom, Jugoslovanski mednarodni simpozij za računalniško tehnologijo in probleme informatike, Informatica 83, Ljubljana 1983, str. 14-18

/2/ Colnarič, M., Rozman, I.; Simulacija multiprocesorskega računalnika s skupnim vodilom, Jogoslovensko savetovanje o mikroprocesorskim sistemima MIPRO 83, Rijeka 1983, str. 2.32 - 2.38

/3/ Rozman, I., Colnarič, M., Bonačič, D.: Analiza učinkovitosti arhitekture s skupnim vodilom pri statistično neenakem zasedanju vodila, Jugoslovensko savjetovanje o mikroprocesorskim sistemima MIPRO 84, Rijeka 1984, str. 3.80 - 3.84

/4/ VMEbus Manufacturers Group: VMEbus Specification Manual, Rev. B, August 1982

/5/ Gustavson, D.B.: Computer Busses - A Tutorial, IEEE MICRO, August 1984, str. 7 - 22

/6/ Taub, D. M.: Arbitration and Control Aquisition in the Proposed IEEE 896 Futurebus, IEEE MICRO, August 1984, str. 28 - 41

/7/ Fischer, W.: IEEE P1014 - A Standard for the High - - Performance VME Bus, IEEE MICRO, February 1985, str. 31 - 41

EFFICIENCY OF MULTIPLE BUS STRUCTURE

I. Rozman, M. Colnarič, B. Stiglic*
TEHNIŠKA FAKULTETA MARIBOR
* ISKRA AVTOMATIKA LJUBLJANA

UDK: 681.3.02

ABSTRACT - Analysis of efficiency two or more buses linked withabus linker is shown in this article. A queueing theory is used. Analysis exactly valid only for exponential distributions for both λ and μ. It is shown how the linking of two buses influences the mean bus response time in comparision with the architecture with consists of one bus and the same number of computers that are connected to two buses.

INTRODUCTION

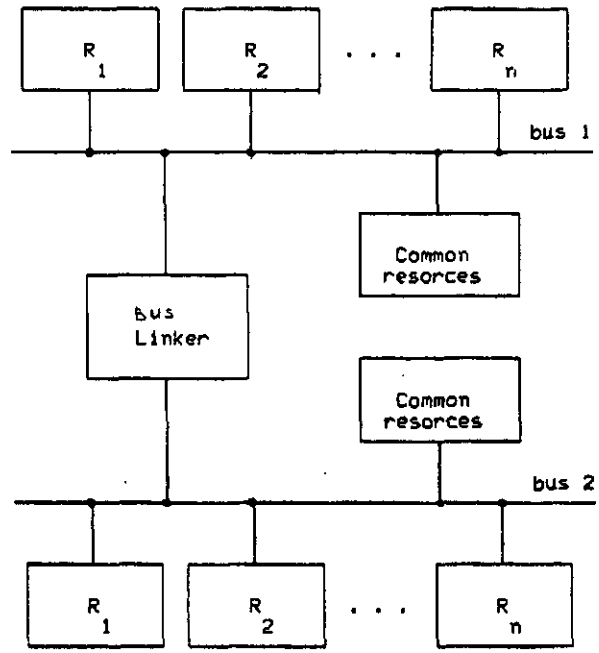
Analysis of efficiency of multicomputer architectures with a common bus is well know in literature /1/, /2/, /3/, /4/. A model of these architectures is derived. An anlytical treatment of this model is based on a queueing theory or better on already derived equations for mean time exsistence in the system which is described by the queue M/G/1/N. With the aid of introduced approximations is shown that results obtained without major tolerance are valid also in cases where distributions are unexponential i. e. in such cases which can be treated by a queue. For this queue an exact mathematical solution is not known. This statement is also valid for the calculation of throughput /but not for the mean bus response time W_N/ in the cases where the arbiter is not FCFS. But in all these cases, individual processors which are connected on a common bus perform a statistically equal work. In literature /3/ is shown the approximation which transforms a model with a statistically unequal work into a model with a statistically equal work.

The problem of conneting two or more buses where computers are connected to each bus still remains an open question. The most important problem is how the linking of two buses influences the mean bus responce time in comparision with the arhitecture which consists of one bus and of the same number of computers that are connected to two buses.

DESCRIPTION OF ARCHITECTURE

The linking of two buses to each other is made for the following reasons:

- the increase of the throughput of the whole architecture is greater if we add one more bus with connected computers;
- the realization price for a bus linker is low in coparison with the price of the whole system;
- the fault tolerance is existing.



Picture 1: Two-Bus-Linking

Bus linker is an active interface which enables two-direction communication between two buses. In its inherent structure, it must contain a memory with enough capacity. Into this memory computers write messages for the computers which are located on the other bus. A too

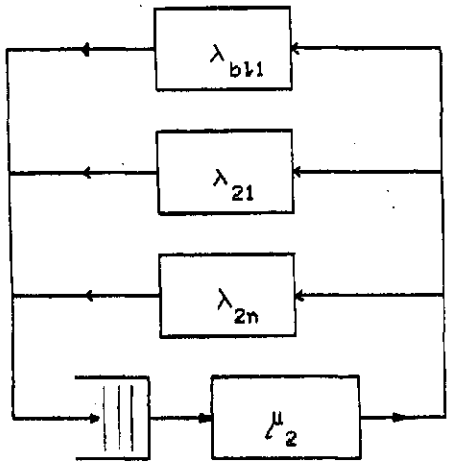
small memory causes an increase of the response time beacuse the messages have to wait to obtain bus and also have to wait that the memory is empty. Therefore it is convenient that the memory is great enough for a two-side transmission in order to permit the transmission of all computers from one bus to the computers in the other bus in the same moment. The bus linker also packs messages into a block. When the bus linker obtains another bus, it permits the transmission of all existing messages in the block to the computers to which the messages are addressed.

THE MODEL

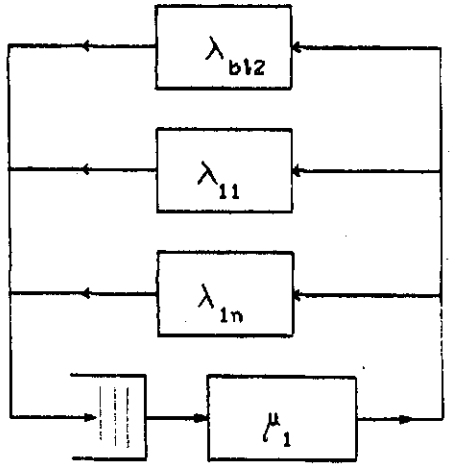
Searching for the suitable model the following suppositions are made:

- bus linker acts undependently for each direction of transmission;
- the time which is necessary for the transmission of message through the bus linker is short in comparisson with the bus occupation time. Therefore it can be neglected;
- the arbiter at each bus is FCFS;
- the bus linker acts in the sense of bus occupation always then when the message enters its empty memory. In cases there are still messages in the memory (the bus linker is waiting for bus occupations), the new message is loaded into the memory.

On the basis of the above mentioned suppositions and the queueing theory the model is formed.



Picture 2(a): The model for transmission of messages in the following directions: bus 1 - bus 2



Picture 2(b): The model for transmission of messages in the opposite direction

In the model shown on picture 2 the bus linker is devided into two parts - separately for each direction. The influence of one bus upon the other is expressed by the source which generates the bus occupations λ_{BL1} or λ_{BL2} according to the direction of transmission. $\lambda_{11} \cdots \lambda_{1n}$ are processors which generate bus occupations for bus 1 with the mean time between bus occupations $\frac{1}{\lambda 1i} \cdot \mu$ presents a bus with the mean bus occupation time $\frac{1}{\mu_1}$. The same impretertation is valid also for bus 2.

THE ANALYSIS

In the analysis of the model only exponential distributions are taken into account for both λ and μ. Other distributions cannot be taken into account exactly. In such cases some approximation methods should be applied. The exponential distribution leads to the solution of the quene Mi/Mi/1/N.

The mean time of retardance in the system is solved by Ferdinand /5/,/6/.

$$u_i = \frac{\lambda_i}{u_i} \tag{1}$$

$$Z_N = \sum_{d_1 \cdots d_i \cdots d_n} \left(\sum_{k=1}^{N} (1-d_k) \right) ! \cdot \prod_{k=1}^{N} \mu_k^{1-d_k} \tag{2}$$

$$d_k = \begin{cases} 1, & \text{request from source k waiting for or being serviced} \\ 0, & \text{request source k is in operational state} \end{cases} \tag{3}$$

$$W_i = \frac{1}{\mu_i} + (1 - \frac{U_N}{L_q}) \cdot \sum_{k=0}^{N} \frac{1}{\mu_k} \cdot u_k \cdot \frac{d}{du_k} \ln Z_N$$

$$U_N = 1 - \frac{1}{Z_N} \qquad (4)$$

$$L_q = \sum_{i=1}^{N} u_i \cdot \frac{d}{du_i} \ln Z_N$$

The calculation Wi according to the equation (4) is difficult. When the value N is high, the calculation of Z_N is not simple. In the calculation Wi the derivation $\ln Z_N$ is necessary to be calculated which additionaly complicates the whole procedure.

Ferdinand /5/, /6/ presents the efficiency of each element $U_N^{(i)}$ as a probability that i^{th} element is not waiting for or being serviced:

$$U_N^{(i)} = Z_{N-1}^{(i)} / Z_N \qquad Z_{N-1}^{(i)} = (Z_N)_{u_i = 0} \qquad (5)$$

$U_N^{(i)}$ can be expressed with the following expression:

$$U_N^{(i)} = \frac{\frac{1}{\lambda i}}{\frac{1}{\lambda_i} + W_i} \qquad (6)$$

from which the mean bus responce time Wi for the element i can be derived (7).

$$W_i = \frac{\frac{1}{\lambda_i}(1 - U_N^{(i)})}{U_N^{(i)}} \qquad (7)$$

If we wont to calculate the throughput of the whole architecture according to the equation (8),

$$T_P = \sum_{i=1}^{N} \frac{1}{W_i + \frac{1}{\lambda_i}} \qquad (8)$$

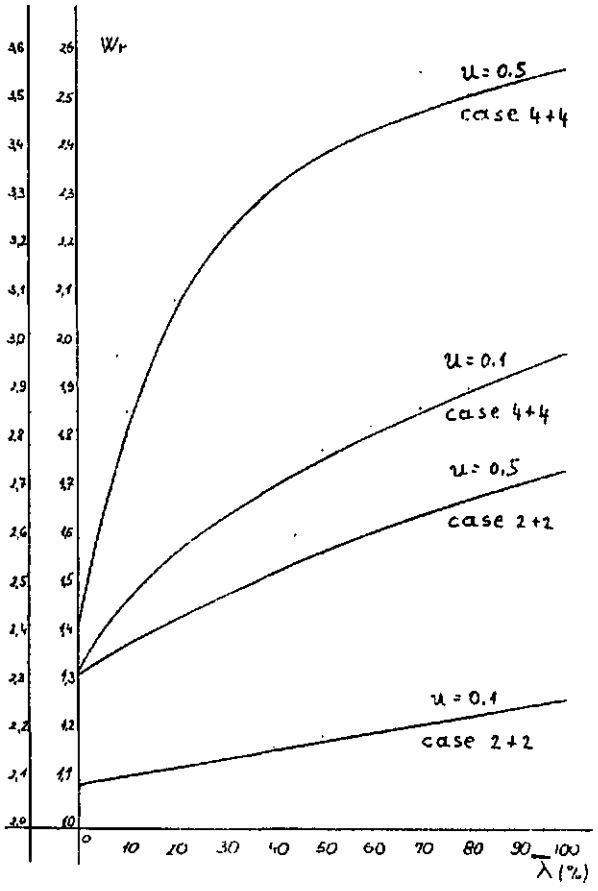
the correct result will not be obtained. The equation (8) is valid only in such cases in which each task obtains the bus. In our case a part of tasks is concluded on the level of one bus. The equation (8) involves only those tasks which occupy another bus but does not include the local ones.

THE NUMERICAL RESULTS

In the presentation of a numerical calculation two examples are shown (picture 3). They clearly present all features which are typical for a link with two buses. In the first example two computers are connected to each bus. In the second example four computers are connected to each bus.

The parameters u_1, i = 1 ... 4 are equal in the first example.

The parameters u_1, i = 1 ... 8 in the second example are equal, too.



Picture 3: Wr vs. % $\overline{\lambda}$

Wr is normative value of the mean bus responce time of the bus:

$$W_r = \mu \cdot W_i \qquad (9)$$

$\overline{\lambda}$ presents the mean arrival of demands in one bus:

$$\overline{\lambda} = \lambda_{(N - L)} \qquad (10)$$

In the equation (10), L stands for the mean time of retardance in the system for the queue M/M/1/N. This parameter can be simply calculated.

From the picture 3 we can see that the increase of traffic through the bus linker approximately parabolically prolongs Wi. The parabolicity becomes more sharp with the saturation.

If we compare Wi for one processor in the two-bus architecture with that in one bus architecture with that in one bus architecture, un-

der the same conditions, we notice that Wi is always smaller in the two-bus architecture. In the first case, in the two bus architecture, when u_i, i = 1 ... 4 = 0,1, λ = 0,1 is w_r = 1,090 while w_r =1,320 in one bus arhitecture.

In the second case we can notice a similar difference in favour of two-bus architecture. This difference becomes more stressed in higher density of traffic (higher u_i). This difference is caused by the packing of messages into a block.

CONCLUSION

From the results obtained we can conclude that the link of two or more buses is especially effective in such cases where one bus becomes saturated. Also the price for the bus linker is not so high that it cannot justify the realization of the linker. The price is approximately 5 % of the price of the whole architecture. When we deal with a very highly coupled system, one bus can be devided into several buses which are connected by bus linkers. This leads to the so called cluster architecture.

REFERENCES

/1/ Rozman, I., Colnarič, M.: Model multiprocesorske/multiračunalniške arhitekture s skupnim vodilom, Jugoslovensko savetovanje o mikroprocesorskim sistemima MIPRO 83, Rijeka 1983, str. 2.109-2.111.

/2/ Colnarič, M., Rozman, I.: Simulacija multiprocesorskega računalnika s skupnim vodilom, Jugoslovensko savetovanje o mikroprocesorskim sistemima MIPRO 83, Rijeka 1983, str. 2.32-2.38.

/3/ Rozman, I., Colnarič, M., Bonačič, D.: Analiza učinkovitosti arhitekture s skupnim vodilom pri statično neenakem zasedanju vodila, Jugoslovensko savetovanje o mikroprocesorskim sistemima MIPRO 84, Rijeka 1984, str. 3.80-3.84.

/4/ Rozman, I., Colnarič, M.: Modeliranje MP/MC arhitektur s skupnim vodilom, Jugoslovanski mednarodni simpozij za računalniško tehnologijo in probleme informatike, Informatica 83, Ljubljana 1983, str. 14-18.

/5/ Ferdinand, A.E.: A Statistical Mechanical Approach to Systems Analysis, IBM J. Res. Develop. Sept. 1970, pp 539-547.

/6/ Ferdinand, A.E.: An Analysis of the Machine Interference Model, IBM Sistem J., No 2, 1971, pp 192-202.

HARDVERSKO POBOLJŠANJE KOD MIKROPROCESORA RADI REALIZACIJE INTERLIVINGA ADRESA PRI PARTITIVNOM ALOCIRANJU MEMORIJE U MULTIPROCESORSKOM SISTEMU

Asim Smailagić
Elektrotehnički fakultet
Sarajevo, Jugoslavija

UDK: 681.3.012

U radu je dato rješenje i potrebna hardverska podrška u procesorskoj jedinici za dekodiranje instrukcija, za u literaturi otvoreni problem generisanja fizičkih adresa kod multiprocesorskih sistema sa partitivnim alociranjem memorije. Izvedeno rješenje omogućuje da mikroprocesor može da prati promjenu stepena interlivinga pojedinog programskog posla, te lociranje i dobavljanje instrukcija i operanada posla u slučajevima kada je slijedeća instrukcija udaljena nekoliko memorijskih modula od tekuće instrukcije programa, što su glavni problemi koje nameće partitivno alociranje memorije sa interlivingom adresa u podskupu memorijskih modula koji su alocirani datom poslu.

HARDWARE IMPROVEMENT IN MICROPROCESSORS FOR THE IMPLEMENTATION OF ADDRESS INTERLEAVING IN MULTIPROCESSOR SYSTEMS WITH PARTITIONED MEMORY ALOCATION SCHEME. This paper presents the solution and required hardware support in the instruction decoding unit of a processor for an open research problem which is related to the generation of physical addresses in multiprocessor systems with partitioned memory alocation scheme. The derived solution enables that a microprocessor can adjust its address mapping mechanism to cope with the changing degree of interleaving of given job, and also locating and fetching of program's instructions and operands when the next instruction might be several modules away from the current instruction. These are the main problems imposed by the partitioned memory alocation scheme with address interleaving in the subset of memory modules which are alocated to a given job.

1. UVOD

U arhitekturi multiprocesorskih sistema glavna memorija je primarni resurs sistema, kojeg zajednički koriste svi procesori. Pošto više procesora u sistemu upućuje simultane zahtjeve za memorijom, to se dijeljenjem glavne memorije u module omogućuje istovremeni pristup ka više modula. Takodje, korištenjem memorije sa interlivingom adresa prema donjim adresnim bitovima, kod koje se sukcesivne adrese smještaju u različite memorijske module, postiže se veća rasutost memorijskih referenci po modulima i veći broj zaposlenih memorijskih modula, što znači i veći memorijski propusni opseg i poboljšanje performansi sistema. Pod stepenom interlivinga se podrazumijeva broj memorijskih modula u čijim adresama je izvršen interliving datog programskog posla.

Kod partitivnog dodjeljivanja memorije programskom poslu se alocira podskup od ukupnog broja memorijskih modula u sistemu (Slika 1), pri čemu od veličine datog posla zavisi koliko će modula sadržavati taj podskup, tako da posao ekskluzivno zauzima dodjeljene mu module, odnosno ne koristi ih zajednički sa drugim poslovima. Pošto su poslovi medjusobno odvojeni kod ovakvog alociranja memorije, to pri njihovom izvršavanju neće dolaziti do konfliktnih memorijskih zahtjeva [1]. Nadalje, pozitivne osobine partitivnog alociranja memorije su da kvar nekog memorijskog modula pogadja samo jedan posao u memoriji; zatim pogodnost za proširenje sistema i održavanje, pošto dodavanje ili uklanjanje nekog memorijskog modula ne utiče na funkcionisanje sistema u cjelini.

Alternativna strategija alociranja memorije, u odnosu na partitivno, je distribuirano alociranje (Slika 2), kod kojeg programski poslovi zajednički koriste sve raspoložive memorijske module u sistemu, pri čemu se svaki posao distribuira preko svih modula [2] . U adresama svih memorijskih modula se obavlja interliving svakog posla. Dakle, dolazi do zajedničkog korištenja svakog memorijskog modula od strane više programskih poslova. To dovodi do konfliktnih zahtjeva od strane dva ili više procesora za istim memorijskim modulom. Zatim, uočljivo je da kvar jednog memorijskog modula pogadja cijeli sistem. S druge strane, kod distribuiranog alociranja memorije postiže se bolja iskorištenost raspoloživog memorijskog prostora, i ne javljaju se posebni problemi u vezi adresiranja u uslovima korištenja interlivinga adresa.

2. ANALIZA PROBLEMA

Da bi se u multiprocesorskom sistemu sa partitivnim alociranjem memorije koristio interliving adresa i tako povećao memorijski propusni opseg, kao mjera performansi sistema, potrebno je naći rješenje za slijedeća dva problema vezana za generisanje fizičkih adresa:

1) Broj memorijskih modula koji se alocira pojedinom poslu varira u zavisnosti od veli-

čine datog posla. To znači da će stepen interlivinga biti različit za pojedini posao. Sa strane procesora to znači da će oni morati imati takav mehanizam preslikavanja adresa koji je u stanju da prati promjenu stepena interlivinga.

ii) Memorijski moduli koji su alocirani nekom poslu općenito ne moraju da budu susjedni jedan drugome. To otežava lociranje instrukcija i operanada programskog posla, jer slijedeća instrukcija može da bude udaljena nekoliko modula od tekuće instrukcije programa.

Interliving adresa u sistemu sa partitivnim alociranjem memorije se može izvesti samo u onim memorijskim modulima u koje je sam smješten dati posao.

Navedimo jedan aspekt problema koji utiče na način interlivinga adresa posla. Neka je dužina programskog brojača L bita. Ako se vrši interliving adresa nekog posla u raspoložive memorijske module, recimo u c modula, morati ćemo biti u stanju obavljati operaciju "količnik-ostatak" od c (Quotient-Remainder) na ovih L bita (označimo ovu operaciju $QR_c(L)$), da bi našli odgovarajući modul i riječ za datu adresu. Vrijednost c zavisi od veličine datog posla. To bi značilo da svaki procesor treba da ima hardverski dio za obavljanje QR_c operacija za sve moguće vrijednosti c, ukoliko bi se interliving adresa posla obavljao na poznati način. To bi impliciralo potrebu ugradjivanja nekoliko QR krugova unutar svakog procesora radi dekodiranja adresa, što bi bilo veoma skupo rješenje. Prema tome, potrebno je pronaći neki drugi način interlivinga adresa programskog posla u sistemu sa partitivnim alociranjem memorije.

3. METOD RJEŠENJA I GENERISANJE FIZIČKIH ADRESA

Naše rješenje opisanog problema se bazira na slijedećem: Ako programski posao zahtjeva toliki broj memorijskih modula koji je stepen od dva, ili broj modula za koji postoji hardverska jedinica za obavljanje QR operacija (npr. za tri modula), onda se interliving adresa posla izvodi na poznati način. U protivnom, podijelićemo memorijske module u toliki broj grupa koji je stepen od dva, a svaka grupa će imati isti broj modula, takav da postoji QR hardver za tu veličinu grupe. Na primjer, neka procesor može da obavlja QR_3 operaciju i neka dati posao zahtijeva šest memorijskih modula. Podijelićemo ove module u dvije grupe, sa po tri modula u svakoj grupi (Slika 3). Ako je, medjutim, za smještanje nekog posla potreban broj modula koji nije djeljiv sa tri, tom poslu će se dodijeliti dodatni memorijski prostor da bi se zadovoljio navedeni uslov djeljivosti. Za odredjivanje grupe koristi se posljednjih g bitova logičke adrese, te ćemo ove bitove nazivati "bitovi grupe". Ako se imaju samo dvije grupe, g će biti jednako 1. Na ovaj način se ustvari postiže dvostruki interliving, tj. ne samo interliving sukcesivnih adresa u različite memorijske module, nego i njihov interliving u različite grupe modula. Na Slici 3. prikazan je interliving adresa posla koji zauzima šest memorijskih modula (6-modulni posao), pri čemu se za indikaciju grupe koristi posljednji bit. Pošto se dobiju dvije grupe sa po tri modula, ova šema se može skraćeno referencirati kao "3-3 interliving". Na ovaj način postigli smo potpuni interliving posmatranog posla preko svih modula koje on zauzima, iako ne raspolažemo odgovarajućim QR_6 krugovima.

Na Slici 4. dat je logički dijagram rješenja preslikavanja adresa za sistem sa partitivnim alociranjem memorije. Registar logičke adrese (RLA) sadrži logičku adresu koju želimo transformisati, a krajnja fizička adresa će se nalaziti u registru fizičke adrese (RFA). Hardverski dio izmedju dva registra obavlja ovu transformaciju. Fizička adresa se sastoji iz dva dijela: broja stvarnog memorijskog modula x i adrese riječi w unutar modula. Prvih (lijevih) L-g bitova programskog brojača, na Slici 4., puni se u šift registar gdje će se obaviti QR operacija. Pošto varijabla g zavisi od broja formiranih grupa modula po poslu, ovdje ćemo iskoristiti šift registar za pomjeranje logičke adrese g bita udesno. Ukoliko je g jednako nuli, onda se cijeli sadržaj registra logičke adrese prebacuje u šift registar, bez pomjeranja. Prema tome, dužina šift registra treba, takodje, da bude L bita. Sada se obavlja QR operacija nad sadržajem šift registra. Ostatak ove operacije će ukazivati na modul unutar grupe, a ovaj isti ostatak zajedno sa bitovima grupe će davati logički broj modula koji tražimo. Količnik će davati tačnu adresu unutar modula. Opišimo kako obavljamo QR operaciju.

Neka c predstavlja bazu QR operacije. Podijelimo šift registar u dvije polovine, od kojih će svaka imati n=L/2 bitova. Označimo sadržaje desne i lijeve polovine sa a i b, respektivno, te se sadržaj šift registra može izraziti kao $a + b2^n$. Nadalje, dekomponujmo b na uc + v, tj. b = uc + v. Ostatak i količnik QR_c operacije se tada nalaze kao:

$$\text{ostatak} = (a+b2^n) \bmod c$$
$$= \lfloor a \bmod c + (b2^n) \bmod c \rfloor \bmod c = \lfloor r_1+r_2 \rfloor \bmod c$$

$$\text{količnik} = (a+b2^n)/c = \left\lfloor \frac{a}{c} \right\rfloor + \left\lfloor \frac{b2^n}{c} \right\rfloor + \left\lfloor \frac{r_1+r_2}{c} \right\rfloor =$$
$$= \left\lfloor \frac{a}{c} \right\rfloor + u2^n + \left\lfloor \frac{v2^n}{c} \right\rfloor + \left\lfloor \frac{r_1+r_2}{c} \right\rfloor$$

gdje je $r_1 = a \bmod c$, $r_2 = (b2^n) \bmod c$,
/ predstavlja cjelobrojno djeljenje, $\lfloor x \rfloor$ je najveći cijeli broj manji ili jednak x.

Dato rješenje uključuje korištenje ROM memorija, što je efikasnije od upotrebe kombinacionih kola za obavljanje QR operacije i odredjivanje količnika, a posebno za veće vrijednosti L. Na Slici 4. se može vidjeti:

- ROM 2 sadrži rezultat od $r_1 = (a \bmod c)$, a ROM 5 sadrži rezultat od $r_2 = \lfloor (b2^n) \bmod c \rfloor$. Izlazi ova dva ROM-a idu na ROM 7, koji će sadržavati $(r_1+r_2) \bmod c$. Na izlazu ROM-a 7 se dobije ostatak, koji zajedno sa bitovima grupe daje logički broj modula.
- riječi u ROM-u 1 će sadržavati rezultat od $\lfloor \frac{a}{c} \rfloor$ uz postojanje jednog bita koji indicira da li su bitovi od a svi jedinice ili ne.
- ROM 3 sadrži u, koje se šiftuje n bita ulijevo.
- ROM 4 će sadržavati vrijednost trećeg člana desne strane jednačine količnika, tj. $\lfloor \frac{v2^n}{c} \rfloor$
- Izlazi ROM-a 2 i ROM-a 5, tj. r_1 i r_2 idu kao ulazi u ROM 6, koji će kao rezultat dati $\lfloor \frac{r_1+r_2}{c} \rfloor$. Pošto vrijedi nejednakost $0 \leqslant r_1+r_2 < 2c$, slijedi da $\lfloor \frac{r_1+r_2}{c} \rfloor$ može biti 0 ili 1. To znači da je izlaz ROM-a 6 dug samo 1 bit.

Sabirač ima ukupno n bitova i obavlja sabira-

nje $\left\lfloor \frac{a}{c} \right\rfloor$, $\left\lfloor \frac{v2^n}{c} \right\rfloor$ i $\left\lfloor \frac{r_1+r_2}{c} \right\rfloor$. Ova tri člana su dugi n-1, n i 1 bit, respektivno. Četvrti član sa desne strane jednačine količnika, tj. $u2^n$, puni se u inkrementator, koji vrši povećavanje za 1 ako sabirač generiše prenos. Razlog zbog čega koristimo n-bitni sabirač i n-bitni inkrementator, umjesto 2n-bitnog sabirača, proističe iz načina sumiranja ovih članova. Naime, na Slici 5. prikazana je dužina svakog pojedinog člana i odnos tih dužina prilikom sumiranja. Kako se vidi, na u jedino može da utiče eventualni prenos kojeg mogu producirati preostala tri člana. Prema tome, za lijevih n bitova potreban je samo jedan inkrementator. Sa Slike 4. se vidi da je za obavljanje QR operacije potrebno najviše dva ROM ciklusa i jedan ciklus sabiranja. To je općenito manje od vremena većine aritmetičkih operacija.

Kako memorijski moduli koje okupira pojedini posao ne moraju biti medjusobno susjedni, to je potrebno transformisati naprijed dobijeni broj logičkog modula u broj fizičkog modula. To se obavlja preko Tabele mapiranja (preslikavanja) modula, koja radi kao baferska memorija. Pošto se dobije broj fizičkog modula, on se dodaje adresi riječi unutar modula i tako se formira krajnja fizička adresa.

U slučajevima kada je za smještanje posla potreban broj memorijskih modula koji je stepen od dva, tada ovu QR operaciju nije potrebno obavljati, pošto se broj modula i adresa riječi unutar modula mogu dobiti na jednostavniji način. Logička adresa se podijeli na dva dijela, pri čemu donjih l bitova (Sl. 4) indicira broj modula kojeg će Tabela mapiranja modula koristiti za dobijanje broja fizičkog modula. Gornjih L-l bitova se pune direktno u registar fizičke adrese. Dva multipleksera (MUX) odabiru koji će se rezultat koristiti.

Jedini nedostatak opisane strategije je što ponekad dovodi do neiskorištenja memorije. Na primjer, ako možemo da obavljamo QR_3 operaciju, a neki posao zahtijeva za smještanje pet memorijskih modula, tom poslu će se morati alocirati šest modula i onda koristiti način interlivinga posla prikazan na Sl. 3. Tabela 1. prikazuje stvarni broj memorijskih modula koji se dodjeljuje pojedinom poslu, broj modula koji bi bio dovoljan u odnosu na veličinu posla, način interlivinga, broj potrebnih bitova grupe g, broj bitova l za slučaj kada posao zahtijeva broj modula koji je stepen od dva, te u posljednjoj koloni indikacija da li dolazi do neiskorištenosti dijela memorijskog prostora. Razlika izmedju druge i prve kolone daje broj neiskorištenih memorijskih modula, čije postojanje indicira posljednja kolona u Tabeli 1. Kao što se vidi iz ove Tabele, 5-modulni i 7-modulni poslovi dobijaju šest i osam modula, respektivno. Tako se dešava da dio memorijskog prostora (jedan modul) bude neiskorišten. Ako su veličina pojedinog memorijskog modula i sastav poslova datog radnog opterećenja sistema takvi da najveći broj poslova iz tog radnog opterećenja zahtjeva do 8 memorijskih modula (što je u realnim sistemima najčešći slučaj), onda problem neiskorištenog memorijskog prostora nije izražen. U slučaju, pak, 9-modulnog, 10-modulnog i 11-modulnog posla svakom od njih se alocira 12 modula, da bi dati posao bio smješten na način 3-3-3-3, tj. moduli se podijele u četiri grupe, pri čemu svaka grupa sadrži po tri modula. Ovdje je za problem neiskorištenog memorijskog prostora rješenje ugradjivanje u procesore QR_5 krugova.

4. ZAKLJUČAK

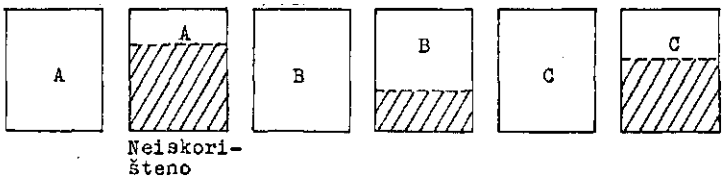
Pošto smo istakli i objasnili u čemu se sastoji problem izvodjenja interlivinga adresa u multiprocesorskom sistemu sa partitivnim alociranjem memorije, u radu smo prezentirali naše rješenje ovog istraživačkog problema i potrebnu hardversku podršku u procesorskoj jedinici za dekodiranje instrukcija. Dali smo deljan opis načina na koji ovo rješenje razrješava problem generisanja fizičkih adresa i prati promjenu stepena interlivinga pojedinog programskog posla.

5. REFERENCE

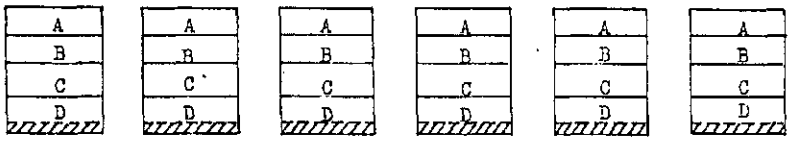
1. Quatse,J. i dr., "A modular computer system and its external access network", AFIPS National.Computer Conf., 1982.
2. Wulf,W.A.,Levin,R.,Harbison,S.P., C.mmp: An experimental computer system, McGraw-Hill, New York, 1981.
3. Smailagić,A., "Using PDP-11 in building mulmultiple processor systems- a critical hardware assesment", IEEE Electrical Electronics Conf., Toronto, October 1981.

Veličina posla u modulima	Broj potrebnih modula	Interliving	g	l	Neiskorištenost memorije
1	1	1	0	0	Ne
2	2	2	0	1	Ne
3	3	3	0	0	Ne
4	4	4	0	2	Ne
5	6	3-3	1	0	Da
6	6	3-3	1	0	Ne
7	8	8	0	3	Da
8	8	8	0	3	Ne
9	12	3-3-3-3	2	0	Da
10	12	3-3-3-3	2	0	Da
11	12	3-3-3-3	2	0	Da
12	12	3-3-3-3	2	0	Ne

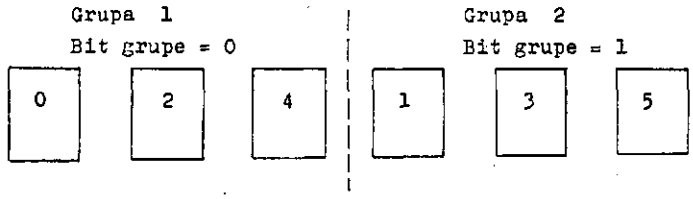
Tabela 1. Broj potrebnih memorijskih modula i način interlivinga za pojedine veličine posla



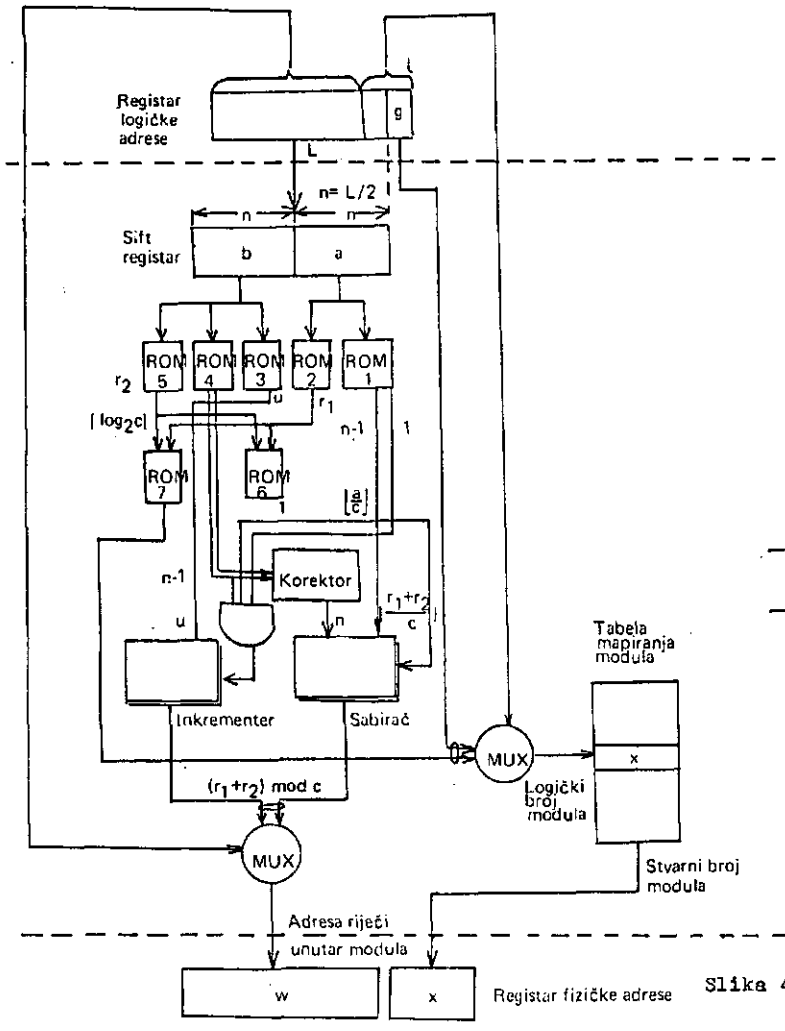
Slika 1. Sistem sa partitivnim dodjeljivanjem memorije, A,B i C su poslovi, kosim linijama je naznačen neiskorišten prostor u memorijskom modulu

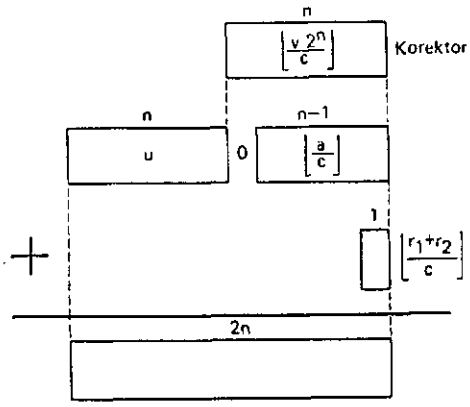


Slika 2. Sistem sa distribuiranim dodjeljivanjem memorije A,B,C i D su poslovi; predstavljanje na nivou riječi



Slika 3. Interliving posla koji zauzima šest memorijskih modula





Slika 5. Dužina pojedinih komponenata kod računanja količnika

Slika 4. Prikaz preslikavanja adresa

IMPLEMENTACIJSKE ZASNOVE ZA IZBOLJŠANJE ZANESLJIVOSTI DELOVANJA POLPREVODNIŠKIH POMNILNIŠKIH SISTEMOV

D. Peček, R. Murn, B. Kastelic, Institut Jožef Stefan
Ljubljana

UDK: 681.3:325.6.0 8

V referatu so opisane nekatere implementacijske zasnove za povečanje zanesljivosti delovanja polprevodniških pomnilniških sistemov. V prvem delu je podan prikaz nekaterih najbolj značilnih detekcijsko korekcijskih postopkov. V nadaljevanju članka pa so opisani postopki za dodatno povečanje zanesljivosti delovanja pomnilniških sistemov.

FAULT-TOLERANT DESIGN TECHNIQUES FOR SEMICONDUCTOR MEMORY APPLICATIONS
In the paper some design aproaches are presented for minimizing the effect of chip(s) or controll logic failures through the use of EDC techniques and through the enhancements to convential error checking and correction facilities.

UVOD

Razvoj pomnilnih integriranih vezij je prekoračil mejo, ki razmejuje LSI in VLSI integracije. Pospešen razvoj bo povzročil, da bodo potrebe po zanesljivih pomnilnih elementih še bolj narastle, pomnilni moduli pa bodo morali biti še bolj neobčutljivi na napake.
Opisane postopke lahko razdelimo v dve skupini: EDC postopki in dodatni postopki za povečevanje zanesljivosti delovanja pomnilnih modulov. Univerzalne rešitve za upravičenost njihove uporabe in način izbora ni mogoče določiti. Za vsako aplikacijo je potrebno posebaj pretehtati najbolj ugodno možnost.

POSTOPKI ZA DETEKCIJO IN KOREKCIJO NAPAK

Hiter razvoj pomnilnih integriranih vezij je povzročil, da so se vzporedno z njihovo uporabo razvijali tudi postopki za povečevanje zanesljivosti delovanja pomnilnih modulov, katerih kapaciteta danes že presega nekaj milijonov zlogov.
V polpreteklem obdobju je bil najbolj uporabljen princip za povečanje zanesljivosti uporaba parnostnega bita ter njegove izvedenke, tako imenovane ortogonalne paritete. Oba postopka omogočata odkrivanje enojne in vseh lih kratnih napak, s pomočjo ortogonalne paritete pa lahko enojno napako tudi popravimo. Postopek zahteva precej dodatne materialne opreme, popravljanje pa lahko izvedemo samo s pomočjo sistemske programske opreme.
Pomemben korak naprej v razvoju postopkov za korekcijo in detekcijo napak je bil storjen, ko so se pojavili tako imenovani Hamingovi kodni sistemi, katerih osnova je minimalna razdalja med kodnimi besedami. S pomočjo standardnih elementov serije 74IS in 74S je možno zgraditi vezje, ki lahko odkriva enojne in dvojne napake in enojne napake tudi popravlja. Postopki so izvedeni s pomočjo materialne opreme in za svoje delovanje ne potrebujejo programske podpore.
Na trišču so se kmalu pojavila integrirana vezja neodvisnih proizvajalcev, s katerimi lahko na dokaj enostaven način izvedemo korekcijo napak v računalniški besedi. Tako lahko izbiramo med vezji AM2960, MC68540, MB1412A, 74LS630/31 in družinami firme Intel in National.

Za uporabo SEC-DED kodnih sistemov (odkrivanje in popravljanje enojne napake, odkrivanje dvojne napake) je najprimernejša taka organizacija pomnilnika, kjer so uporabljeni pomnilni elementi, ki imajo samo en izhod. Statistika pojavljanja napak je takšna, da je izredno malo verjetno, da bi se istočasno pojavili dve napaki v računalniški besedi, saj sta obe verjetnosti statistično neodvisni.
Problem nastopi, ko hočemo uporabljati SEC-DED kodne sisteme in moramo iz kakršnega koli razloga uporabljati pomnilna vezja z več izhodi. V tem primeru je verjetnost pojavitve večkratne napake precej večja kot v prejšnjem primeru. SEC-DED postopki se izkažejo za neprimerne. Moramo vpeljati tako imenovane SEC-DBD kodne sisteme, kar pomeni : detekcija in korekcija enozložnih napak ter detekcija dvozložnih napak. Za takšne kodne sisteme ne obstajajo komercialno dobavljiva integrirana vezja, razviti pa so algoritmi, ki jih je mogoče preliti v materialno opremo.

S povečanjem kapacitete pomnilnih modulov pada njihova zanesljivost, če ne poskrbimo za naslednje korake pri povečanju zanesljivosti. To so tako imenovani DEC-TED kodni sistemi, ki so sposobni popravljati tudi dvojno napako, ter odkrivati trojne napake. Nekateri takšni kodni sistemi so danes že teoretično dokazani, možno jih je tudi izvesti z materialno opremo,ki pa seveda ni tako preprosta,kot je v primeru SEC-DED sistemov.

Ko se pri snovanju pomnilnega modula odločamo o tipu uporabljenega kodnega sistema, je potrebno določiti tudi vse postopke in akcije, ki nam omogočajo spremljanje in vrednotenje napak, kot tudi njihovo obdelavo. Zato je potrebno že na samem začetku skrbno določiti povezavo in način komunikacije z centralnim procesorjem. Univerzalne rešitve ni, pri odločanju pa nas mora voditi stopnja zanesljivosti, ki jo želimo doseči ter hitrost delovanja v primeru stalnih napak. Vendar to še ni vse. Narava pojavljanja napak je takšne, da se lete počasi kopičijo (trenutne napake zaradi sevanja alfa delcev). Zato je potrebno na nivoju

sistemske programske opreme zagotoviti občasen pregled vseh vitalnih pomnilnih lokacij. Pogostost pregleda lahko določimo eksperimentalno, z modelom in simulacijo ali pa računsko.

Detekcijsko korekcijski kodni sistemi pa niso edini način, kako povečati zanesljivost pomnilnih sistemov.
Odmislimo skrbnost zasnove stikalnega načrta krmilne logike in pomnilnega polja, kvaliteto izdelave tiskanega vezja in uporabljenih komponent.
Zanesljivost sistema lahko precej povečamo z naslednjimi prijemi:

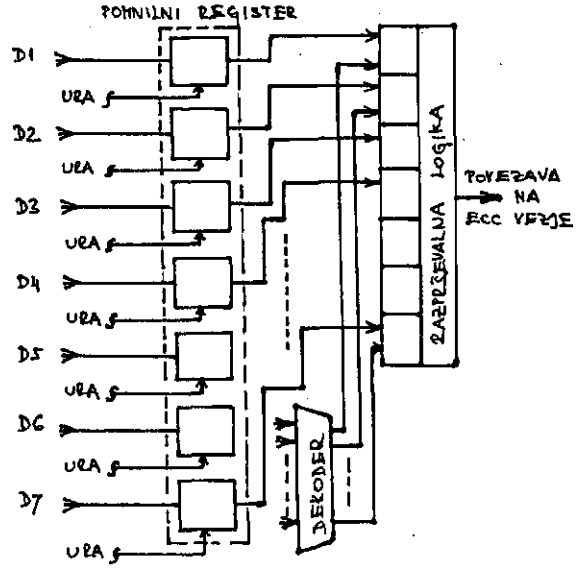
- razporejanje podatkov
- pasivna redundanca
- pricip dvojne nagacije
- zapovrstna korekcija

Osvetlimo vsak pojem posebaj:

Razporejanje podatkov

Razporejanje podatkov je način, s pomočjo katerega zmanjšamo vpliv okvarjenih pomnilnih elementov.
Za razporejanje podatkovnih bitov uvedemo tako imenovano razprševálno vezje, ki poskrbi, da se izhodi ene okvarjene računalniške besede preslikajo v več računalniških besed. Postopek je izveden s pomočjo materialne in programske opreme. Enostavno povedano, gre za neke vrste transplantacijo dobrih in okvarjenih podatkovnih bitov. Če vsebuje neka beseda N okvarjenih bitov, moramo poiskati N dobrih (neokvarjenih) računalniških besed in v vsako vsaditi en okvarjen bit, oziroma ga zamenjati. Tako smo N kratno napako v eni besedi razbili v N enkratnih napak v N besedah.
Poseben primer razporejanja podatkov je tako imenovano prenaslavljanje. V tem primeru želimo okvarjene računalniške besede združiti v čim manjše - seveda neuporabno pomnilno področje.

Slika 1 prikazuje princip realizacije razporejanja podatkov.



Slika 1. Razporejanje podatkov

Pasivna redundanca

Pasivna redundanca omogoča zamenjavo okvarjenega pomnilnega pomnilnega področja brez posega operaterja. Ta pristop lahko uporabimo za same pomnilne elemente in za nadzorna vezja. Zaradi zmanjšanja obsega materialne opreme se običajno odločimo, da pasivna redundanca do zamenjave ne opravlja funkcije sledenja. Zato je ob zamenjavi še poskrbeti da se podatki prenesejo iz okvarjenega dela pomnilnika v njegovo zamenjavo. Tudi za to akcijo moramo predvideti funkcije v sistemski programski opremi.

Princip dvojne negacije

Dvojna negacija je postopek izveden s pomočjo programske opreme, s pomočjo katerega lahko popravimo dvojno napako v okviru SEC-DED kodnega sistema. Oglejmo si postopek na zgledu. Osnova bodi dvojna napaka, ena stalna in ena trenutna.

1 0 1 0 1 0 1 0 1	previlen podatek
S T	stalna in trenutna napaka
1 0 0 0 0 0 1 0 1	nepravilen podatek
0 1 1 1 1 1 0 1 0	negiramo in vpišemo
0 1 0 1 1 1 0 1 0	preberemo
1 0 1 0 0 0 1 0 1	negiramo
1 0 1 0 1 0 1 0 1	vpis pravilne vrednosti
1 0 0 0 1 0 1 0 1	enojna napka pri čitanju, sistem jo popravi

Zaporedna korekcija

Zaporedna korekcija je postopek, ki ga izvedemo s pomočjo programske opreme. Osnova je zopet SEC-DED kodni sistem. S pomočjo zaporedne korekcije povečamo učinkovitost SEC-DED kodnega sistema. Postopek sloni na zgodovini pojavljanja stalnih enojnih napak. Zaradi akumulacije napak lahko pride do pojavitve stalne dvojne ali večkratne napake, ki je s standardnim SEC-DED sistemom ne moremo odpraviti.
Za zaporedno korekcijo moramo shranjevati sindrome okvarjenih pomnilnih lokacij po vrstnem redu pojavljanja. Ko ECC odkrije nepopravljivo napako, se prejšnji sindromni biti uporabijo za rekonstrukcijo osnovne besede. Tudi ta metoda zahteva dodatno sistemsko programsko opremo.

ZAKLJUČEK

Smo v točki razvoja, ko bo računalnik prevzel veliko opravil, ki danes še temeljijo na neposrednih, medčloveških odnosih. Če mu ne bo uspelo, da bi se prijazno nasmehnil ali pa nas grdo pogledal, naj vsaj svoje delo opravlja brez napak.

SLOVSTVO

Blahut, R. E. Theory and Practice of Error Control Codes, A-Wesley. 1983
C.L.Chen Error Correcting Codes for Semiconductor Memory, IBM J. Res. Develop, vol.28, March 84

MODEL PROCESORA ZA OBRADU I RASPOZNAVANJE SLIKA

Slobodan Ribarić
VTA, Zagreb

UDK: 681.327

U radu je prikazana hijerarhija procesora u sustavima za obradu i raspoznavanje slika. Na temelju svojstva koja procesori pokazuju u računanju aritmetičkih $E\langle n\rangle$ i logičkih izraza $L\langle n\rangle$, te izvođenju operacija nad slikovnim elementima, procesori se razvrstavaju u četiri razine: H_0 - procesori koji izvršavaju samo binarne i unarne operacije, H_1 - procesori koji izvršavaju m-arne operacije nad jednovrsnim aritmetičkim ili logičkim izrazima, H_2 - procesori sa svojstvima "računskog demona" i konačno, procesori razine H_3 sa svojstvima "računskog demona" koji izvršavaju operacije nad slikovnim elementima. Procesori razine H_3 predstavljaju glavnu komponentu modela visokoparalelnog računala R koji je opisan u drugom dijelu rada.

A PROCESSOR MODEL FOR PICTURE PROCESSING AND RECOGNITION: This paper presents a processor hiearchy in the systems for picture processing and recognition. Regarding the properties of processors in evaluation of the arithemtic $E\langle n\rangle$ and logical expressions $L\langle n\rangle$, and the performance of the operations on the pixels they are classified into four levels: H_0 - processors that perform only binary and unary operations, H_1 - processors that perform m-ary operations on uniformed arithemtic or logical expressions, H_2 - processors with the properties of the "computational demon", and, finally, processors of the Level H_3 with the properties of the "computational demon" which perform operations on the pixels. Processors of the level H_3 are the main component of a model of highly parallel computer R described in the second part of the paper.

1. UVOD

U posljednjih desetak godina zanimanje za područje obrade i raspoznavanja uzoraka u stalnom je porastu. Razvoj VSLI tehnologije i rad na rješavanju problema saobraćanja čovjek - stroj, u okviru projekta slijedeće generacije računala, snažno je pospješio istraživanja i primjenu novih metoda i tehnika na području raspoznavanja uzoraka.

Obrada i raspoznavanje slika predstavlja u većini slučajeva veliki računski problem koji zahtijeva veliku performansu računala. Na primjer, potrebna performansa računala (izražena u milijunima instrukcija u sekundi) za tipičnu obradu slike rezolucije 512x512 slikovnih elemenata, u stvarnom vremenu iznosi između 65 i 650 MIPS-a [1]. U ovom kontekstu pod obradom slike u stvarnom vremenu podrazumijevamo obradu u vremenu manjem od vremena potrebnog za obnavljanje slike na zaslonu ($< 4 \cdot 10^{-2}$ s).

Obrada i raspoznavanje slika, s druge strane, predstavlja problem s visokim stupnjem inherentnog paralelizma u podacima i operacijama. To nam omogućava postizanje visokih performansi upotrebom multiprocesorskih sustava, matričnih procesora, sistoličkih polja i računala upravljanih tokom podataka [1], [2].
Procesori - procesni elementi važna su komponenta paralelnog sustava. Oni u zavisnosti od svojih sposobnosti obrade gotovo izravno utječu na performansu sustava [3].
U prvom dijelu rada dane su definicije procesora u sustavima za obradu i raspoznavanje slika. Prikazan je njihov utjecaj na broj potrebnih koraka T za računanje aritmetičkih $E\langle n\rangle$ i logičkih izraza $L\langle n\rangle$. Procesori su razvrstani u četiri hijerarhijske razine. U drugom dijelu rada opisan je model visoko paralelnog računala R koji je poslužio kao osnova za izvedbu mikroprocesorskog klasifikatora rukom pisanim znakova [4].

2. HIJERARHIJA PROCESORA

Procesore prema njihovim svojstvima u računanju aritmetičkih i logičkih izraza razvrstavamo u četiri razine:

a) Procesor hijerarhijske razine H_0

Aritmetički izraz E možemo definirati kao pravilno tvoren niz sastavljen najmanje iz jedne od četiri aritmetičke operacije (+, -, *, /), lijevih i desnih zagrada (ukoliko su potrebne) i atoma koji su konstante ili varijable [3].
Aritmetički izraz sastavljen iz n atoma označavat ćemo sa $E\langle n\rangle$.
Logički izraz L je pravilno tvoren niz sastavljen najmanje iz jedne logičke operacije, lijevih i desnih zagrada (ukoliko su potrebne) i atoma koji su logičke varijable ili konstante.
Logički izraz L sastavljen iz n atoma označavat ćemo sa $L\langle n\rangle$.

Tvrdnja 1

Upotrebom potrebnog broja procesora sa ulazima samo za dva atoma (procesor izvršava binarne operacije $a \circ b$ ili unarne operacije) aritmetički izraz $E\langle n\rangle$ može se izračunati u

$$T[E\langle n\rangle] \geq \lceil \log_2 n \rceil \text{ koraka,}$$

gdje $\lceil x \rceil$ označava gornju cjelobrojnu vrijednost broja x. Dokaz tvrdnje 1 je trivijalan [3].

Tvrdnja 2

Broj potrebnih koraka za računanje logičkog izraza $L\langle n\rangle$ uz pomoć procesora s ulazima za samo dva atoma je:

$$T[L\langle n\rangle] \geq \lceil \log_2 n \rceil .$$

Definicija 1

Procesor koji izvodi osnovne aritmetičke operacije i logičke operacije, te ima ulaze za dva operanda i zadovoljava tvrdnje 1 i 2 naziva se procesor p_0 ili procesor hijerarhijske razine H_0.

Većina današnjih procesora u računarskim sustavima su procesori hijerarhijske razine H_0. U biti, procesor p_0 je procesor u današnjim uniprocesorskim sustavima (von Neumannovim računalima).

b) Procesor hijerarhijske razine H_1

Procesor hijerarhijske razine H_1 izvršava m-arne operacije nad jednovrsnim aritmetičkim i logičkim izrazima.

Jednovrsni aritmetički izraz $E_u\langle n\rangle$ je pravilno tvoren niz sastavljen samo iz jedne vrste aritmetičkih operacija (ili +, ili -, ili *, ili /), lijevih i desnih zagrada (ukoliko su potrebne) i n atoma koji su konstante ili varijable. Na sličan način može se definirati jednovrsni logički izraz: $L_u\langle n\rangle$ je pravilno tvoren niz sastavljen iz jedne vrste logičkih operacija, lijevih i desnih zagrada (ukoliko su potrebne) i n elemenata koji su logičke konstante ili varijable.

Tvrdnja 3

Procesor p_1 neka je takav da ima m ($m > 2$) ulaza (procesor izvršava m-arne operacije). Upotrebom potrebnog broja procesora p_1 aritmetički izraz $E_u\langle n\rangle$ izračunava se u:

$$T[E_u\langle n\rangle] \geq \lceil \log_m n \rceil \quad \text{koraka.}$$

Tvrdnja 4

Jednako vrijedi i za računanje jednovrsnog logičkog izraza $L_u\langle n\rangle$. Broj potrebnih koraka je:

$$T[L_u\langle n\rangle] \geq \lceil \log_m n \rceil .$$

Definicija 2

Procesor koji izvršava jednovrsne aritmetičke i logičke operacije i ima $m>2$ ulaza za atome (izvršava m-arne operacije), te zadovoljava tvrdnje 3 i 4 naziva se procesor p_1 ili procesor hijerarhijske razine H_1.
Primjer izvedbe procesora hijerarhijske razine p_1 dan je u radu [6] gdje je opisan procesor za istovremeno zbrajanje m n-bitnih podataka.

c) Procesor hijerarhijske razine H_2

Procesor hijerarhijske razine H_2 predstavlja neku vrstu "računskog demona" [7].

Definicija 3

Procesor p_2 ili procesor hijerarhijske razine H_2 opisan je trojkom:

$$p_2 = (U, I, \mu),$$

gdje U je skup raspoloživih atoma aritmetičkog ili logičkog izraza, I je skup rezultata aritmetičkih ili logičkih operacija koje se izvršavaju nad raspoloživim atomima, i μ je funkcija:

$$\mu : U \to I.$$

Funkcija μ predstavlja skup aritmetičkih i logičkih operacija koje se izvode u jednom koraku bez obzira na tip operacije (jednovrsna ili raznovrsna) i broj operanada n. Primjeri realizacije procesora p_2 su funkcionalna memorija [8] i programibilni RAM u računalu za obradu slika. Cytocomputer [5].
Procesori u hijerarhijskim razinama H_0 - H_2 imaju takve značajke da vrijedi slijedeći odnos:

Ako neki procesor p ima svojstva procesora hijerarhijske razine H_2, tada taj procesor ima i svojstva procesora hijerarhijske razine H_1 i hijerarhijske razine H_0.

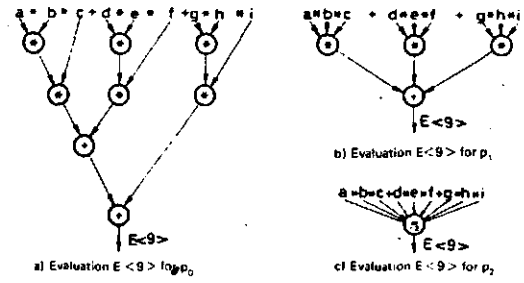
Primjer 1

Neka je procesor p iz hijerarhijske razine H_2. Ograničimo mu funkciju μ samo na jednovrsne operacije i neka je $m > 2$. Tada procesor poprima svojstva procesora razine H_1. Ako za ovaj procesor zahtijevamo $m = 2$ dobivamo procesor hijerarhijske razine H_0.

Primjer 2

Slika 1 prikazuje računanje aritmetičkog izraza E 9 za procesore pojedinih hijerarhijskih razina. Broj potrebnih procesora i broj koraka iznosi (sl. 1):

3 procesora	p_0	$T[E\langle 9\rangle] = 4$
3 procesora	p_1	$T[E\langle 9\rangle] = 2$
1 procesor	p_2	$T[E\langle 9\rangle] = 1$



a) Evaluation $E\langle 9\rangle$ for p_0
b) Evaluation $E\langle 9\rangle$ for p_1
c) Evaluation $E\langle 9\rangle$ for p_2

Slika 1.

d) Procesor hijerarhijske razine H_3

Procesor hijerarhijske razine H_3 upotrebljava se za obradu slika.

Definicija 4

Dvodimenzionalna slika je graf f^* funkcije f:

$$f^* = \{((a_j, a_k), v) \mid (a_j, a_k) \in A,\ v = f(a_j\ a_k)\},$$

gdje je $A = \{(a_j, a_k) \mid 1 \leq a_j \leq r_1,\ 1 \leq a_k \leq r_2;\ r_1, r_2 \in N\}$; N je skup prirodnih brojeva, $v \in R$, R je skup realnih brojeva, a funkcija f je restrikcija funkcije $\varphi : N^2 \to R$:

$$f = \varphi \mid A.$$

gdje je N^2 skup uredenih dvojki prirodnih brojeva.

Fizikalna pozadina funkcije f i definicije slike nalazi se u procesu digitalizacije slike - gdje se upotrebljava uzrokovanje da bi se iz analogne slike dobio skup realnih brojeva, čije su vrijednosti u procesu kvantizacije pretvorene u n diskretnih vrijednosti [9].
U mnogim praktičnim primjenama - funkcija φ je definirana kao

$$\varphi : N^2 \to Z,$$ gdje je Z skup cijelih brojeva.

U definiciji 4 element grafa funkcije (a_j, a_k), v) naziva se slikovni element (pixel), a r_1 i r_2 su dimenzije slike. Definicija slike razlikuje se od uobičajenih definicija [9]-[11].

U slikovnom elementu $((a_j, a_k), v)$ može se, u skladu s operacijama koje se izvršavaju, vrijednost slikovnog elementa v smatrati operandom, a prva komponenta (a_j, a_k) može se promatrati kao ishodišna, odnosno odredišna adresa operanda.

Definicija 5

Procesor hijerarhijske razine H_3 opisan je trojkom:

$$p_3 = (U, J, o),$$

gdje je U skup grafova funkcije f^* (skup ulaznih dvodimenzionalnih slika), J skup grafova funkcije g^* (skup izlaznih - obradenih dvodimenzionalnih slika). Slikovna operacija o preslikava U u J:

$$o : U \to J.$$

Procesor hijerarhijske razine H_3 izvršava preslikavanje o u jednom koraku.
Procesor p_3 izvršava operacije nad slikovnim elementima $((a_j, a_k), v) \in f^*$ tako da djeluje na prvu komponentu (a_j, a_k) i na drugu komponentu v.

Primjer 3

Prikažimo operaciju zbrajanja dvaju dvodimenzionalnih slika f_1^* i f_2^* i uvećanje rezultirajuće slike za dva puta:

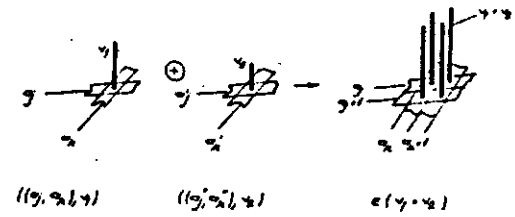
$$((a_j, a_k), f_1(a_j, a_k)) \oplus ((a_j', a_k'), f_2(a_j', a_k')) =$$

$$= \varepsilon (f_1(a_j, a_k) + f_2(a_j', a_k'))$$ za sve

$$(a_j, a_k) \in A_1,\ (a_j', a_k') \in A_2;\quad A_1 = A_2,$$

gdje je ε pravilo koje definira podskup parova A_3 (rezultirajuće slike) nad kojima će se dignuti rezultirajuća vrijednost $(f_1(a_j, a_k) + f_2(a_j', a_k'))$. U našem slučaju (sl. 2) je to:

$$\{(a_j, a_k), (a_j, a_k + 1), (a_j + 1, a_k), (a_j + 1, a_k + 1)\}$$



Slika 2.

Slikovana operacija o može biti i operacija pomaka slikovnih elemenata (slika), operacija ispitivanja jednakosti slikovnih elemenata, zbrajanje, oduzimanje, množenje, dijeljenje slikovanih elemenata (slika) i operacije s konstantom, te logičke operacije nad binarnim slikama.

Procesor p_3 sve operacije (jednostavne i mješane) nad raspoloživim slikovnim elementima (ili cijelim slikama) izvršava u jednom koraku.

Iako procesor p_3 ima svojstva "računskog demona" [7] svoju fizikalnu pozadinu, djelimično, može naći u optičkoj obradi slika [10], gdje se Fourierova transformacija slike ili križna korelacija dvaju slika izvršava u jednom koraku.

3. MODEL RAČUNALA R

Model visoko paralelnog računala R razvijen je u skladu s značajkama modela za tipične faze raspoznavanja: pretprocesiranje, izlučivanje karakteristika i klasifikacija [12]. Kao osnova modela sustava za obradu i raspoznavanje slika uzet je paralelno-serijski konus raspoznavanja koji su predložili L. Uhr i R Douglass [13].
Bitni element modela računala predstavlja procesor hijerarhijske razine H_3 - procesor za obradu slika.

Definicija 6

Model računala R je *šestrorka:*

$$R = (F_0, P_3, M, C, S, F_k),$$

gdje su F_0 i F_k skupovi ulaznih, odnosno izlaznih slika (prema definiciji 4).

Slika iz skupa početnih slika nastala je paralelnim unosom slike (npr. osvjetljavanjem transparentnog materijala i projekcijom na polje fotosenzitivnih elemenata). Slika iz skupa konačnih slika predstavlja izlaz iz sustava.

P_3 je skup hijerarhijske razine p_3 (Definicija 5):

$$P_3 = \{ p_{31}, p_{32}, \dots, p_{3k} \} .$$

M je skup memorijskih ravnina za pohranjivanje početnih slika, konačnih slika i međuslika koje nastaju u fazi obrade (u skladu s paralelno - serijskim konusom). Memorijske ravnine se u načinu djelovanja značajno razlikuju od konvencionalnih memorija von Neumannovog modela koje imaju sekvencijalni pristup:
U slučaju kada je memorijska ravnina izabrana - svi podaci, koji su pohranjeni u njoj, su istovremeno raspoloživi. U slučaju upisa u memorijsku ravninu - svi podaci se istovremeno upisuju.

C predstavlja skup upravljačkih funkcija:

$$C = \{ c_1, c_2, \dots, c_n \} ,$$

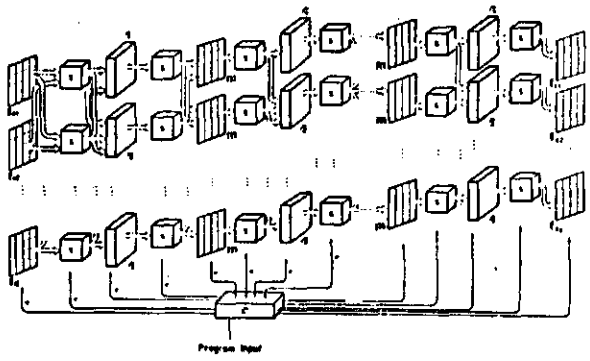
koje se upotrebljavaju za upravljanje procesorima (aktiviranje, definiranje operacija), memorijskim ravninama (izbor i određivanje operacija), prospojnim modulima (aktiviranje, određivanje smjera toka podataka).

Prema načinu upravljanja obradom računalo može biti računalo s upravljačkim tokom, računalo upravljano tokom podataka ili njihova kombinacija.

Skup $S = \{ s_1, s_2, \dots, s_p \}$ predstavlja skup prospojnih modula koji usmjeravaju rezultate izvršavanja slikovnih operacija prema memorijskim ravninama i procesorima hijerarhijskog nivoa H_3.

Računalo R je visokoparalelno računalo koje omogućava istovremenu obradu na skupu ulaznih slika F_0. Struktura $m\text{-}s\text{-}p_3\text{-}s$ odgovara strukturi protočnih segmenta [15] tako da računalo ima značajke MISD arhitekture računala: istovremeno su aktivni svi protočni segmenti na različitim međuslikama.

Model računala opisan u ovoj sekciji znatno se razlikuje od von Neumannove standardne arhitekture [14], ali bez obzira na visoku razinu njegove apstrakcije, postoji niz računala koje imaju neke njegove značajke.
Slika 3 prikazuje model visoko paralelnog računala R.



Slika 3.

4. ZAKLJUČAK

U radu je prikazana hijerarhija procesora koji se upotrebljavaju u sustavima za obradu i raspoznavanje slika. Na temelju paralelno-serijskog konusa raspoznavanja predložen je model visoko paralelnog računala R. Bitna gradevna komponenta modela je procesor hijerarhijske razine H_3. U skladu s modelom računala R razvijen je paraleini mikroprocesorski klasifikator uzoraka čiji je jedan modul opisan u [4].

5. LITERATURA

1 S. Ribarić, Računari upravljani tokom podataka, Informatica, br. 4, 1982. str. 3-11.

2 K. S. Fu (ed), VLSI for Pattern Recognition and Image Processing, Springer - Verlag, Berlin, 1984.

3 D. J. Kuck, A. Survey of Parallel Machine Organization and Programming, ACM Comp. Surveys, vol 9, str. 29-60, mart, 1977.

4 S. Ribarić, N. Pavešić, Mikroprocesorski klasifikator numeričkih znakova s mrežom memorijskih komponenti LSI, Informatica 2/3, 1983. str. 162-167.

5 P. E. Danielsson, S. Levialdi, Computer Architectures for Pictorial Information System, Computer, vol 14, no. 11, novembar 1981., str. 53-67.

6 N. Konvaras, et al., A. Digital System for Simultaneous Addition of Several Binary Numbers, IEEE Trans, Comput., vol. C-17, str. 992-997, oktobar 1968.

7 O. G. Selfridge, Pandemonium: A Paradigm for Learning, u Pattern Recognition (ed. L. Uhr), John Wiley & Sons, New York, 1966.

8 P. L. Gardner, Functional Memory and Its Microprogramming Implications, IEEE Trans. on Comput., vol. C-20, No. 7, str. 764-775, juli 1971.

9. A. Rosenfeld, A.C. Kak, Digital Picture Processing, Academic Press, New York, 1976.

10 A. Rosenfeld, Picture Processing by Computer, Academic Press, New York, 1969.

11 T. Pavlidis, Algorithms for Graphics and Image Processing, Springer - Verlag, Berlin, 1982.

12 S. Ribarić, Paralelizem v razpoznavanju dvodimenzionalnih vzorcev in njegov upliv na model procesorja, doktorska disertacija, Fakulteta za elektrotehniko, Ljubljana, 1982.

13 L. Uhr, R. Douglass, A Parallel - Serial Recognition Cone System for Perception: Some Test Results, Pattern Recognition, vol. 11, str. 29-39, 1979.

14 A. W. Burks, et al., Preliminary Discussion of the Logical Design of an Electronic Computing Instrument, u G.G. Bell et al. (ed.), Computer Structures: Readings and Examples, HcGraw-Hill, New York, 1971.

15 C. V. Ramamoorthy, H.F. Li, Pipeline Architecture, ACM Computing Surveys, vol, 9, mart, 1977., str. 61-101.

REALIZACIJA RAČUNALNIŠKE LOGIKE S CELIČNIMI STRUKTURAMI

Veselko GUŠTIN
Andrej DOBNIKAR
Fakulteta za elektrotehniko
Ljubljana, Tržaška 25
Yugoslavia

UDK: 681.3.5.7

POVZETEK - V referatu omenjamo realizacijo poljubnega algoritma s podatkovnim tokom. Taka realizacija omogoča predvsem veliko število hkratnih izvajanj. Pokazali bomo tudi primer prevedbe grafa podatkovnega toka v programirano celično polje. Zasledili bomo poseben formalni zapis polja celic, ki ga imenujemo matrika logičnih elementov in povezav. Prav tako so omenjeni nekateri algoritmi za računalniško načrtovanje logičnega polja.

COMPUTER LOGIC REALIZATION THROUGH CELLULAR STRUCTURES

ABSTRACT - The paper outlines the transformation of general computing algorithm into its data flow graph notation which shows the degree of parallelism that is characteristic for the algorithm. The way, how the data flow graphs can be realised by the programmable cellular array is also shown. Special formal description of cellular array is introduced. It is called matrix of logic elements and its connections. Its use in automatic logic design is briefly described.

1. Uvod

Uvajanje mikroprogramiranja v aparaturno opremo računalnika je omogočilo večkratno izkoriščanje aritmetične logične enote (ALE). Prenašanje vsebine preko le-te pa nam je naenkrat odprlo pot v realizacijo poljubno obsežnega algoritma. Na ta način načelno skromen nabor funkcij ALE izvede še vse drugačne operacije. Z večkratnim preslikovanjem vsebine skozi ALE na primer dobimo: normalizacijo števila, pretvorbo celega števila v zapis s plavajočo vejico, množenje, deljenje, sinus kota ipd. Lahko rečemo, da skoraj ni omejitev, kaj vse s preprostim ALE naredimo.

Pa vendar, ali je možno razširiti osnovni nabor funkcij v ALE tako daleč, da sploh ne bi govorili o nekem številu novih funkcij, ampak o sprotni konfiguraciji ALE vezja. In kateri logični gradniki so sposobni oblikovati neko vezje, oziroma preslikavo? Zelo primerna oblika takih vezij so polja celic, ki jih lahko na preprost način programiramo in naredimo takšna kakršna želimo. Omenimo naj še, da je celica logični operator ali pomnilni element. Skoraj ne moremo govoriti o fiksni strukturi, pač pa o strukturi, ki jo sproti prilagajamo potrebam paralelnega procesiranja algoritma.

V nadaljevanju prispevka si bomo pogledali realizacijo algoritma s podatkovnim tokom in prevedbo le-tega v polje celic. Opisali bomo izbiro celice in njene lastnosti utemeljili.

2. Preslikava algoritma v data-flow graf

Prav presenetljivo je, koliko hkratnih akcij lahko opravimo v procesiranju nekega "strogo zaporednega algoritma". Ob pregledu razne literature ugotovimo, da so o paralelnem procesiranju razmišljali že pred več kot 30 leti. To je komaj nekaj za tem, ko je bil narejen prvi računalnik. Kar ne moremo razumeti je, kako da množica teoretičnih pristopov ni našla svoje mesto v kaki konkretni, komercialni realizaciji, temveč je ostalo pri več ali manj laboratorijskih poskusih. Paralelno procesiranje lahko razvrstimo po sledečih načelih:

- Petrijeve mreže
- Diagram računanja (computation graph)
- Paralelna programska shema (parallel program schemata)
- Podatkovni tok (data-flow).

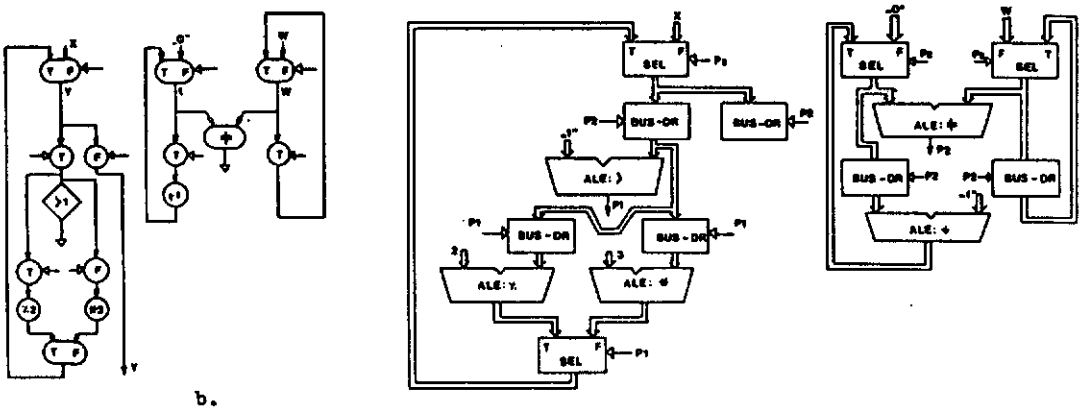
Več se bomo ustavili samo pri podatkovnem toku (data-flow). Poznamo več načinov izvajanja data-flow (DF) grafov. Izbrali bomo tistega, ki je najbolj podoben principom paralelnega proce-

siranja, to je način s p o r o č i l (token mode).

Kaj zmore realizacija nekega algoritma v DF načinu, bomo spoznali, ko preslikamo poljubni zapis algoritma v nekem programskem jeziku (na primer Pascal) v DF graf.

```
input (w,x);
 y := x; t := 0;
 while t<>w do
  begin
   if y > 1 then y :=y/2
            else y :=y*3;
   t := t + 1;
  end;
output y.
```

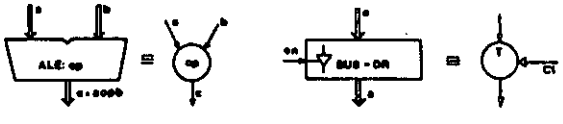
a.



b.

Sl. 1. Algoritem v programskem jeziku Pascal nasproti podatkovnemu toku

Na tem mestu nas bolj kot preslikava jezika Pascal v DF graf zanima, kako lahko DF graf preprosto realiziramo z logičnimi komponentami. Poskus take preslikave srečamo pri /4/. Realizacija po naši zamisli bo nekoliko odstopala od osnovnih načel DF grafov, bo pa vsekakor najbližja zveza med grafom in njegovo logično realizacijo. Če si zamislimo preslikavo osnovnih DF gradnikov v logične gradnike, bi narisali kot primer sledeči zvezi:



Sl. 2. Primera logičnega in DF gradnika

Podobno bi bile videti še vse ostale preslikave DF gradnikov. Kako pridemo do njih? Podatkovne vhode oziroma izhode DF grafa zamenjamo s podatkovnimi vodili. Kontrolni vhodi ali izhodi DF grafa nam v logični shemi predstavljajo kontrolne signale. Operacijo v DF vozlišču pa nadomestimo z aritmetično logično enoto (ALE), ki je sposobna želeno funkcijo izvesti. Če torej podatkovni tok narišemo s tako dobljenimi logičnimi gradniki, dobimo logično shemo vezja (glej sliko 3).

3. Realizacija DF grafov

a) Realizacija DF grafov s komponentami srednje (MSI) in velike (LSI) integracije ni nobena posebnost. Na tržišču se pojavlja cel kup komercialnih vezij srednje integracije (prekodirniki BCD, množilniki, delilniki itd.), ki povezani v smislu DF grafa dajo že kar njegovo realizacijo.

Sl. 3. Realizacija DF grafa

Ni težko ugotoviti, da smo logično shemo preprosto narisali tako, da smo sliko DF grafa uporabili za osnovo. Tako dobljeno logično vezje bi pravilno delovalo, ko bi ga časovno vskladili. Povsod tam kjer smo napisali ALE:op, si mislimo vezja večjih integracij, na primer ROM, RAM ali PAL. Le-te uspešno uporabimo kot prekodirnike z želeno funkcijo. Tak način izgubi svoj smisel, če vsak algoritem posebej pomeni novo vezje. Rešitve moramo torej iskati v drugačnih logičnih vezjih.

b) Začetne poskuse reševanja logičnih funkcij s poljem celic srečamo že pred letom 1970. Najpogostejša ideja je, kako napraviti tako univerzalno (iterativno) celico, ki bo primerna za reševanje nekaj aritmetičnih operacij, na primer seštevanja (odštevanja), množenja in deljenja.

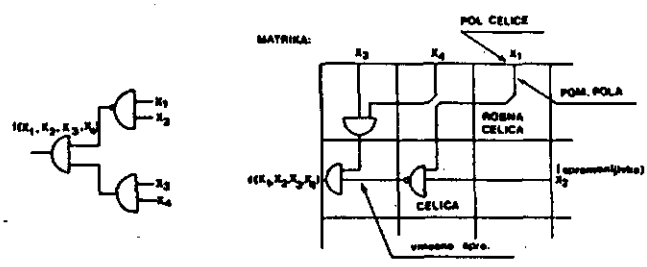
Drugačen primer realizacije srečamo pri /1/, kjer celica ne vrši samo logične funkcije v

smislu logične preslikave od enega vhoda do drugega (iterativno), temveč služi tudi že kot stikalo, pomnilnik ipd. Tisto, kar nas privlači pri celičnih realizacijah, smo srečali šele v /2/.

Celica, ki je tu definirana, predstavlja tako logični operator kot samo stikalo ali pot, preko katere lahko povezujemo druge celice. Avtor je predvidel 16 različnih možnosti celic, ki predstavljajo tako logične funkcije kot tudi povezave med celicami. S takim naborom celic lahko rešimo poljubno obsežno in zamotano s t r u k t u r o logičnega vezja.

Lahko rečemo, da smo celico iz /2/ vzeli za osnovo našemu razmišljanju. Celico na nek način programiramo. Polje programiranih in ponovno programiranih celic uporabimo za izvajanje logičnih funkcij, na primer iterativnih povezav seštevalnika, množilnika ipd. Skup takih aritmetičnih in logičnih operatorjev nam ne predstavlja nič drugega kot podatkovni tok nekega algoritma. Obsežno polje celic - struktura logičnih gradnikov in povezav - pa je sposobna realizirati poljuben algoritem.

Celico si zamišljamo kot pravokotno obliko. Na vse štiri strani je povezana z drugimi prav takimi celicami. Preko teh štirih strani lahko povezujemo ali vhode ali izhode sosednjih celic. Vsebina celic nam predstavlja ali logično funkcijo, na primer AND, OR ali samo stikalo, ne primer od zgornje celice preko naše v desno nam sosednjo celico. Primer celice v matriki logičnih elementov in povezav (MLEP) nam kaže slika 4.



Sl. 4. Celica v MLEP

Opredelimo nekaj pojmov celice in matrike povezav (MLEP): povezave med celicami ali matrikami gredo preko p o l o v ali t r n o v . Ker je logično vezje celice poljubno o r i e n t i r a n o , tedaj tudi pol nima vedno istega p o m e n a . Lahko se obnaša kot vhod ali izhod. Celicam, ki so razmeščene po robovih matrike, pravimo r o b n e celice. Tistim, ki so znotraj matrike in niso robne, rečemo n o t r a n j e celice. Značilno za robne celice je, da iz njih izstopajo ali vstopajo funkcije ali spremenljivke. Med notranjimi celicami se običajno prenašajo v m e s n e spremenljivke. Čemu nam služi pol, ali vhodu ali izhodu, nam pove njegov pomen. Vsi poli skupaj pa nam definirajo funkcijski z n a č a j celice. Skup celic imenujemo polje ali matrika ali m a k r o celica. Če imamo celice razvrščene samo v eni vrstici ali stolpcu, tedaj govorimo o v e k t o r j u celic.

4. Definicija celice qi, j v MLEP

Zamisel celične strukture vidimo na sliki 4. Skušajmo sedaj definirati celico, ki nam predstavlja osnovni gradnik v matriki logičnih elementov in povezav, MLEP. Splošno celico qi,j v MLEP lahko zapišemo kot četverce spremenljivk x/s/ in polov g/s/:

Definicija 1:

(xm/s1/, gm/s1/; xn/s2/, gn/s2/; xo/s3/, so/s3/; xp/s4/, gp/s4/)i,j, kjer je

xm, xn, xo, xp ∈ X,
X = {x1, x2, x3, ..., xm, xn, xo, xp}

ter s1, s2, s3, s4 ∈ {1,2,3,4,}

množica vhodnih oziroma izhodnih spremenljivk, robnih ali vmesnih.

gm, gn, go, gp ∈ G; G = {g1, g2, ..., gm, gn, go, gp, ... };
G = G1 o G2;

G1 = {i | vhod v celico, o| izhod celice, ni| negirani vhod, no| negirani izhod celice, -| pol ni opredeljen, 0| vhoda ni} .

Dodatno potrebno tolmačenje vhodov oziroma izhodov nam omogočajo pojmi iz množice G2:

G2 = {and | logična IN funkcija, or| logična ALI funkcija, exor| funkcija ekskluzivni ALI, q | pomnilna celica, s | indeks vhodnega pola, ki ga vežemo izključno na označbe o ali no pri celicah z več različnimi logičnimi izhodi; s∈ N, ^| vežemo izključno z - ali 0 ali i oziroma ni } .

Pomeni parametra s so sledeči:

S = {1 | levi pol celice, 2 | zgornji pol celice, 3| desni pol celice, 4| spodnji pol celice} .

$i \in N$, $j \in N$, $N = \{1,2,3,\ldots\}$ indeksa i in j sta naravni števili, pri tem je največji indeks iN = I, jN = J. Indeks i označuje vrstico, indeks j označuje stolpec matrike ‖Q‖I,J, katere element je qi,j, torej $qi,j \in$ ‖Q‖I,J.

Zapis logične funkcije je načelno v PDNO, PKNO ali kaki drugi vmesni ali minimalni obliki. Celo nenormalne oblike ne predstavljajo nobene ovire za preslikavo v začetni zapis matrike. V enotno celično strukturo zajamemo eno ali več (mnogopol) logičnih funkcij. Zagotoviti moramo le, da vsak term ni daljši od treh spremenljivk. Vmesne spremenljivke skupaj zopet predstavljajo term, za katerega prav tako velja, da ne sme zajeti več kot tri spremenljivke. Postopek razbijanja v terme zaključimo, ko lahko prvotno logično funkcijo zapišemo z množico termov, ki niso daljši od treh spremenljivk. Preslikava ni definirana za sledeče primere:

- če so več kot trije vhodi v nek logični operator. V tem primeru si torej pomagamo z razbijanjem termov,
- če logična funkcija terma ni v naboru G2, tedaj ali razširimo nabor logičnih funkcij celice ali rešimo logično funkcijo z obstoječim naborom.

Primer 1:

Zapišimo logično funkcijo treh spremenljivk po Def. 1:

$f(x1,x2,x3) = x2.x3' \vee x1'.x3 \vee x1.x2'$.

Velja naj, da po razbijanju v terme dobimo sledeče zveze za robne in vmesne spremenljivke:

$x4 = x2.x3'$; $x5 = x1'.x3$; $x6 = x1.x2'$;

$x7 = x4 \vee x5 \vee x6$.

Sledeči zapis predstavlja enoumno preslikavo logične funkcije v obliko, ki ji pravimo z a č e t n i vektor ali matrika in jo označujemo s ‖Qz‖. Vsak term zapišemo v svojo celico, ki je poljubno orientirana in na poljubnem mestu v vektorju oziroma matriki, kot na primer:

$$Qz = \begin{matrix} (x7,o\ or;x4,i;x5,i;x6,i) \\ (x4,o\ and;x2,i;x3,ni;-,-) \\ (x5,o\ and;x1,ni;x3,i;-,-) \\ (x6,o\ and;x1,i;x2,ni;-,-) \end{matrix}$$

5. Sestav polnega razreda celic

Ločeno bomo obravnavali polnost za

a) logične funkcije in
b) povezave.

Nabor funkcij dvovrednostne logike, ki sestavljajo polni razred, srečamo v /1/ pa tudi v drugih učbenikih, ki obravnavajo realizacijo preklopnih vezij. Bolj nas zanima, kateri je tisti najmanjši nabor povezovalnih elementov, celic, s katerim lahko realiziramo povezave vsakega logičnega vezja. Pomagali si bomo s teorijo grafov, s tistim delom, ki govori o logičnem vezju kot o grafu. Vozlišče predstavlja logični operator, povezave grafa so povezave vhodov in izhodov logičnega vezja. V našem primeru bomo vezje gledali kot graf, ki ga skušamo vpeti v matriko elementov in povezav. Vsaka celica predstavlja kvadrat, ki ima na sredi vsake stranice priključek - pol. Le-ta je lahko ali vhod ali izhod ali pa ni uporabljen. Zamislimo si, da tako vpete povezave realiziramo s celicami, ki zadoščajo vsem zahtevam po povezavah.

Vsi možni slučaji povezav med poli nas vodijo v polni nabor celic. Najprej zapišimo, kaj je polni razred matrike logičnih elementov in povezav.

Definicija 2:

Polni razred povezav sestavlja tak nabor, ki ne zahteva nobenih pogojev glede začetne topologije vezja, niti ni nobenih odvečnih polov, ki bi se jih morali izogibati med povezovanjem in povezovanje teče z izrabo vseh možnosti takega polnega nabora (slika 5a). Vsa odstopanja mimo pravkar omenjenih zahtev, nas vodijo v pogojno polni razred. Zakaj polni razred sestavlja ravno 15 celic? Vseh grafov na štirih točkah je 64, neizomorfnih pa 11. Če torej izluščimo vse smiselne povezave, nam ostane (narisanih) 15!

S tremi izreki bomo pokazali, kateri je najmanjši nabor, ki sestavlja pogojno polni razred:

Izrek 1 Potrebujemo celico, ki lahko poveže dva in dva nasprotna pola tako, da med obema povezavama ni stika.

Dokaz: Lahko najdemo slučaj, ko povežemo celice

$qi1,j1 \leftrightarrow qi2,j2 \leftrightarrow \ldots \leftrightarrow qi,j \leftrightarrow \ldots \leftrightarrow qin,jn \leftrightarrow qi1,j1$

tako, da povezave predstavljajo zaključeno pot, torej obkrožajo polje celic P. Obstajata vsaj en qix,jx, ki je znotraj polja P in vsaj en qixx,jxx, ki je zunaj zaprtega polja P in ju moramo med seboj v ravnini povezati. Tako povezavo lahko izvedemo le tako, da na vsaj enem mestu križamo pot, tako da uporabimo celico iz izreka 1.

Izrek 2 Potrebujemo celico, ki lahko poveže poljubni vhodni pol z vsaj dvema izhodnima.

Dokaz: Ko celica qij krmili več kot dva vhoda v naslednje celice, tedaj potrebujemo vsaj 1 ali več razvejitev signala: vhod pride na poljuben pol in se razcepi na vsaj dva pola. Obema pogojema zadosti celica enega tipa, na primer štiripolno stičišče.

Izrek 3 Potrebujemo celice, ki povežejo vsaj dva bližnja (sosednja) pola, na primer iz leve vodoravne smeri navzgor, tako da omogočajo sestaviti zanko.

Dokaz: Že v izreku 1 smo omenili možnost, da lahko nastopi slučaj, ko so celice povezane v zaključeno pot. To se zgodi vedno takrat, ko imamo opraviti z več kot pa samo drevesnimi strukturami, na primer pri povratnih povezavah avtomatov. Zanke pa lahko realiziramo le, če imamo elemente, ki nam omogočajo zavijanje v vse smeri.

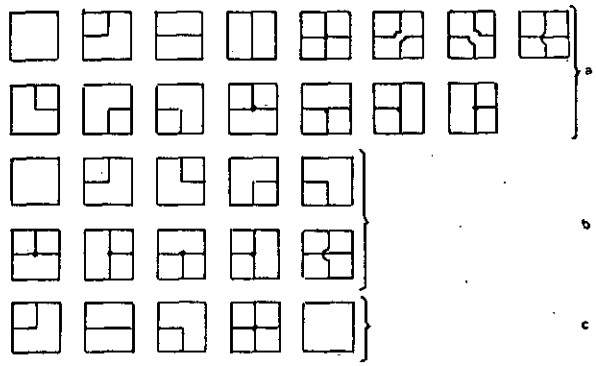
Izreki 1 - 3 nam služijo za ugotavljanje, ali nek nabor celic pripada polnemu razredu. Takoj lahko ugotovimo, da vsem trem izrekom zadostita že dve celici:

a) stičišče vseh štirih polov in
b) stik med dvema in dvema nasprotnima poloma.

Omenjeni nabor predstavlja pogojno polni razred, saj veljajo nekatere omejitve:

- vsako postavljeno stičišče zahteva zase celotno vertikalo in horizontalo;
- signal se ustavi šele ko zadenemo ob vhodno/izhodno celico;
- na stičišču poljubne horizontale in vertikale sme biti le ena celica z logično funkcijo. Če jih je več, tedaj si morajo slediti v smiselnem vrstnem redu, torej izhod gre na vhod naslednje, izhod te zopet naprej, itd.

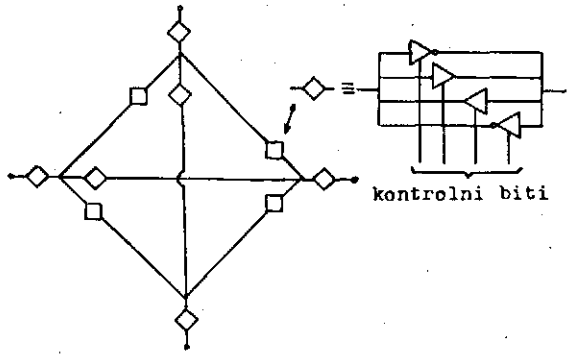
Na sliki 5 vidimo nekaj primerov polnih in nepolnih naborov celic.



Sl. 5. Polni (a), pogojno polni (b) in nepolni (c) razredi povezovalnih celic

6. Realizacija 4 polne kvadratne celice

Za lažje razumevanje celice postavimo tako zgradbo, ki bo omogočala povezave v smislu polnega razreda in ki bo tudi logično polna.



Sl. 6. Realizacija celice

Vezje, ki ga vidimo na sliki 6, nam omogoča logične funkcije V, &, |, ↓ z možnostjo, da nekatere vhode po potrebi tudi negiramo. Število bitov za tako celico dosega že kar nemogočo številko, zato kaže to število zmanjšati do razumnih meja (do 8), vendar tako, da je celica še vedno element polnega razreda. Tudi tehnologija postavi svojo piko na i, saj realizacija z na primer bufferji s tretjim visoko impedančnim izhodom ali odprtim kolektorjem pride v poštev le za TTL, medtem ko si moramo v CMOS-u poiskati drugačne rešitve. Končno pove svoje tudi postavitev na silicijevo ploščico, ko je površina še kako pomembna in stremimo za tem, da je čim manj krmilnih bitov in čim več prostora za logiko oziroma povezave v celici.

Programiranje povezav je smiselno, če uporabimo enega od sledečih načinov:

a. dejansko prežiganje povezav (PROM, PAL),
b. tovarniško programiranje povezav (ROM, gate-array) in

c. laboratorijsko reprogramiranje [illegible] (EPROM).

Naš namen je [illegible] programiranje [illegible] tako, [illegible] izvedemo [illegible], preprosto [illegible]. Ce[illegible] tako, da ji [illegible] funkcijo. [illegible], če k vsaki [illegible] [illegible], bi krili vse [illegible].

Vse do [illegible] razli[illegible] sti matrike [illegible]. Za primer vzemimo, da [illegible]:

a. posamezna celica ali
b. niz celic, vendar ne cela h/v in
c. cela horizontala ali vertikala oziroma v njih.

Primer a se nam zdi najbolj verjeten. [illegible] čakujemo okvar več sosednjih celic, primer [illegible]. Za oba primera se nam porodi zanimiva rešitev; vedeti moramo le, katere so te celice. Za postopke povezovanja jih označimo kot neuporabne. Take celice oziroma nize celic preprosto obidemo in si pot poiščemo mimo njih. Večji problem je izpad c e l o t n e vertikale ali horizontale. V tem primeru ne moremo več govoriti o eni MLEP, ampak več matrikah. Od posameznega primera je seveda odvisno, kaj lahko naredimo s tako presekano matriko: ali jo uporabimo ali pa zavržemo.

7. Algoritmi za prevedbo iz začetne matrike v končno

<u>Analitični</u> pristop temelji na:

a. povezovanju logičnih elementov in
b. povezovanju večjih skupin že povezanih logičnih elementov, makrojev.

Pri obeh tipih povezovanj naletimo na nekatere že znane primere o povezovanju in razmeščanju integriranih komponent na plošči tiskanega [illegible]ja. Fizikalno snovanje in modeliranje nam za makroje povsem ustreza, medtem ko za nivo logičnih operatorjev ni toliko primerno, saj so problemi okrog povezovanja, razmeščanja in orientacije celic precej svojstveni.

Omenimo, katere postopke lahko uporabimo za makro celice in delno za celice:

- vezje, grafični model
- particije minimalnih povezav
- particije minimalnih kasnitev
- postavitveni problem
- iskanje poti ali routing.

Zbrani so v /3/, /5/, zato jih tu ne bi ponavljali, lahko se posvetimo raje algoritmom za prevedbo začetne matrike v končno; realizaciji logičnega vezja iz osnovnih podatkov o funkciji:

a. [illegible] spremenljivk, razvrstitev po števi[illegible]
b. [illegible] ste sprem. navzdol ugotavl[illegible]mo povezavo, če te ni, tedaj c, sicer a (brez vrinjanja),
c. iskanje najbližjih sosednjih vertikal in horizontal,
d. [illegible] in realizacija povezav po del[illegible] algoritmu,
e. [illegible] do zadnje.

[illegible] [illegible]ieren, naj omogoča [illegible] interaktivno vklju[illegible] načrtovalca [illegible]. Kako postopamo? Začetno vezje, [illegible] s poljubno prepletenimi povezavami [illegible] mi skušamo spraviti v matriko našega [illegible] vezav in log. funkcij. Poseben grafični jezik naj bi omogočal naslednje operacije:

a. razmaknitev horizontalnih oz. vertikalnih linij,
b. rotacijo log. elementa,
c. premestitev log. elementa,
d. postavljanje poševnih linij v matriko navpičnih in vodoravnih linij,
e. pomikanje začetka ali konca linije na poljubno mesto (vejanje) in
f. razbijanje več kot štiripolnih log. ali [illegible]kalnih elementov.

Vse začetno podane matrike $\|Qz\|$ nas po transformaciji $\mathcal{T}$ pripeljejo v razred rešitev $S\|Qz\|$. Obstaja rešitev $S'\|Qz\| = \|Qk\|$, ki ji pravimo dobro oblikovana realizacija $\|Qz\|$, za katero velja $S\|Qz\| \subseteq S'(\|Qz\|)$. Za primer ko je $S(\|Qz\|) = S(\|Qz\|) = \|Qk\|$ opt rečemo, da je to najboljša rešitev in $\mathcal{T}$, preslikava pomeni polni razred. Za optimalno rešitev $\|Qk\|$ [illegible] velja:

- [illegible]m manjša matrika $\|Qk\|$ opt in
- [illegible] krajše povezave.

Težko je verjetno, da bi kar takoj zadeli naj ugodnejšo rešitev, pač pa se ji lahko približamo z več poskusi. Zelo pomembno je [illegible], [illegible]ko obrnemo začetne celice in kam jih po[illegible]. Najboljšo rešitev bomo najhitreje našli ob tesnem sodelovanju načrtovalca in računalniške podpore.

8. Zaključek

Delo bomo nadaljevali z realizacijo celičnega polja v CMOS mikroelektronski izvedbi. Kot najbolj ugodna realizacija se je pokazala celica, ki jo vidimo na sliki 6. Vsaka tehnologija, kot smo omenili, ima svoje dobre in slabe strani, svoje omejitve in prednosti. Možnosti, ki jih trenutno imamo, kažejo, da bi tako polje vsebovalo okrog 10 * 10 = 100 celic. Res da to ni prav veliko, je pa zagotovo spodbuden začetek k računalniškim arhitekturam ne von Neumannovega tipa.

9. Literatura

1. J.Virant, Logično snovanje računalniških struktur in sistemov, Ljubljana, 1979.
2. F.B.Manning, An Approach to Slighly Integrated Computer - Maintained Cellular Arrays, IEEE Trans. on Comp., Vol. C-26, No. 6, str. 536-552, 1977.
3. V.Guštin, A.Dobnikar, Interna poročila, 1982-1984.
4. J.B.Dennis, B.Jock, Data-Flow Supercomputers, Computer, Col. 13, 1980.
5. A.D.Friedman, P.R.Menon, Theory & Design of Switching Circuits, Comp. Science Press, 1975.

SINTEZA SPREMENLJIVE ARHITEKTURE RAČUNANJA Z VLSI PROGRAMIRNIM POLJEM NA OSNOVI DF ANALIZE

Andrej DOBNIKAR
Veselko GUŠTIN
Fakulteta za elektrotehniko
Ljubljana, Tržaška 25
Yugoslavia

UDK: 681.519.7

POVZETEK - Prispevek podaja postopke analize podatkovnih tokov za splošni problem računenja in sinteze strukturne rešitve ob izbiri Univerzalnega logičnega polja ULP z možnostjo reprogramiranja funkcij v realnem času.

CHANGEABLE COMPUTING ARCHITECTURE SYNTHESIS WITH VLSI PROGRAMMING ARRAY BASED ON DF ANALYSIS. ABSTRACT - The paper outlines the procedures of data flow analysis of general computing problems and structure design synthesis when Universal Logic Array ULA with the real time reprogrammable posibility is considered.

I. Uvod

Pri iskanju novih konceptualnih rešitev v okviru računalniških sistemov največkrat izhajamo iz dveh predpostavk. Prvič, da novi koncept omogoča boljše performančne lastnosti od obstoječih sistemov in drugič, da njegova realizacija ustreza trendom v tehnologiji, kar pogojuje sprejemljivo ceno.

Veliko večino današnjih računalniških sistemov označuje serijska lastnost procesiranja. Zanje se je udomačilo ime po avtorju, von Neumannov tip računalnika. Osnovna hiba teh sistemov je v počasnosti procesiranja, ki je predvsem posledica "ozkega grla" med procesno enoto in pomnilnikom. Njegova arhitektura je toga in diktira sekvenčni način procesiranja ne glede na naravo problema, katerega rešuje. Alternativne rešitve, ki na osnovi analize podatkovnih poti vnašajo paralelnost v procesiranje, imenujemo arhitekture podatkovnih tokov.

Razvoj tehnologije gre v smeri procesiranja gostote elementov na enoto ploščine ter iskanje rešitev, ki dvigujejo hitrost in nižajo ceno elementa. Zaradi tega so s stališča tehnologije zanimive predvsem takšne strukture, ki izkazujejo visoko stopnjo regularnosti in so hkrati kar najbolj univerzalne.

Definicija 1: Logično polje je univerzalno, če je mogoče z njim realizirati vse različne resurse računalniške logike, oziroma poljubno podmnožico resursov računalniške logike $\mathfrak{R}$:

$$\mathfrak{R} = \langle \text{POV}, \text{KV}, \text{PV}, \text{PP} \rangle \quad , \qquad (1)$$

kjer je:

POV - pomnilno vezje
KV - krmilno (sekvenčno) vezje
PV - procesno (kombinacijsko) vezje
PP - povezovalne poti.

V nadaljevanju bomo predpostavljali, da univerzalno logično polje (ULP) zadošča zgornji definiciji. Primera takšnih polj sta podrobno opisana v /1, 2/. ULP je programirno, če je mogoče spreminjati vsebino polja, s tem pa tudi njegovo funkcijo.

II. DF notacija računanja

Računanje nekega problema lahko podajamo grafično ali analitično. Obe predstavitvi sta enakovredni in prikazujeta strukturo računanja, iz katere je razviden podatkovni in kontrolni del.

1. Grafična predstavitev računanja

Računanje je grafično podano z $g = \langle PG, DG \rangle$, kjer je PG prednostni graf, DG pa podatkovni graf. Zanju velja:

$$PG = \{VV, IV, OR, RO, UP\} \quad , \qquad (2)$$

kjer je:

VV - vstopno vozlišče ($\mathfrak{L}$)

IV - izstopno vozlišče ($)
OR - operatorji računanja (male črke)
RO - relacijski operatorji ($\wedge$(AND), $\vee$(OR), (kretnica))
UP - usmerjene povezave

in:

DG = {PC, OR, UP} (3)

kjer je:

PC - pomnilne celice (velike črke)
OR - operatorji računanja (male črke)
UP - usmerjene povezave

2. Analitična predstavitev računanja

Računanje je podano analitično z $\mathcal{A} = \langle KI, DI \rangle$, kjer pomeni KI krmilne izraze, DI pa podatkovne izraze.

KI = {OR, RO, +, -} , (4)

kjer je:

OR - operatorji računanja (male črke)
RO - relacijski operatorji (.(stik), $\vee$(ALI), &(IN), < > (iteracijski operator), ( ) (navadni oklepaj))
+,- - znaka, ki sledita imenu decizijskega operatorja in določata rezultat odločitve.

Teorem 1: KI je regularni izraz (RI), ki ga realizira kompozicija končnih avtomatov (KA).

Dokaz: V algebri dogodkov nastopajo le operatorji (., V, < >), v KI pa dodatno še operator &. Ker vsak RI realizira en sam KA, sledi, da lahko vsak KI zapišemo kot produkt delnih RI, torej KI = R/1 & R/2 & ... R/N. Tedaj KI realizira ravno N KA, katerih izhodi so povezani z operatorjem &.

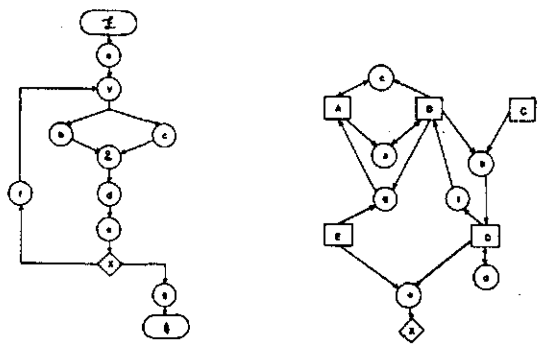
DI = $\{D_i\}$, (5)

kjer je:

D_i = {NVPC, NIPC}
NVPC - niz vhodnih podatkovnih celic operatorja i
NIPC - niz izhodnih podatkovnih celic operatorja i.

Primer 1: Grafični in analitični prikaz računanja R.

A. Grafični prikaz:



Sl. 1. Primer grafične notacije R

B. Analitični prikaz:

KI = a . < (b&c).d.e-.f > .e+.g (6)

DIa = ((A,B),(B))
DIb = ((B,C),(D))
DIc = ((A,B),(A))
DId = ((D),(D))
DIe = ((E,D),(X))
DIf = ((D),(B))
DIg = ((B,E),(A))

III. Analiza paralelnosti računanja na osnovi DF notacije

Za par operatorjev računanja (a,b) velja, da sta lahko v eni od naslednjih odvisnosti:

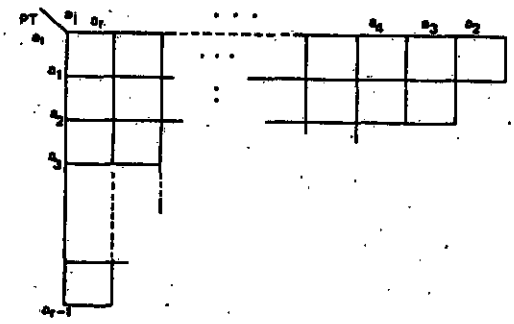
$a \parallel b$ - a,b sta neodvisna, kar pomeni, da je:
$Ia \cap (Vb \cup Ib) = 0$ in:
$Ib \cap (Va \cup Ia) = 0$, kjer je:
Ii-izhodna množica podatkovnih celic operatorja i
Vi-vhodna množica podatkovnih celic operatorja i
$a \wedge b$ - a,b sta paralelna
$a > b$ - a je pred b
$a \curlywedge b$ - a,b sta na alternativnih poteh.

Definicija 2: Če velja $a \not> b$, $b \not> a$ in $a \parallel b$, potem sta a in b paralelna, oziroma $a \wedge b$.

Zgornje odvisnosti v okviru določenega serijskega računanja je mogoče predstaviti v trikotni prednostni tabeli PT na sliki 2. Pri tem velja predpostavka: $a_i > a_j$, če $i < j$.

Definicija 3: Naslednja transformacijska pravila omogočajo uvajanje paralelnih relacij:

1. a.b $\longrightarrow$ $a \wedge b$ če in samo če velja:
$Ib \cap (Va \cup Ia) = 0$ in
$Ia \cap (Vb \cup Ib) = 0$



Sl. 2. Prednostna tabela PT

2. $a \wedge b \longrightarrow a.b$ ali $b.a$
3. $(a \vee b).c \longleftrightarrow a.c \; b.c1$, kjer je c1 kopija c
4. $(a \vee b) \vee c \longrightarrow a \vee b \vee c$
   $a \vee (b \vee c) \longrightarrow a \vee b \vee c$
   $(a \wedge b) \wedge c \longrightarrow a \wedge b \wedge c$
   $a \wedge \quad \longrightarrow a \wedge b \wedge c$

Postopek za določitev maksimalne stopnje paralelnosti za serijsko podano računanje R:

1. Formiranje drevesne strukture S = (s1,s2,..).
   Kontrolnemu vozlišču ustreza izraz, v katerem nastopa na mestu zanke posebno ime operatorja. Vozlišče naslednjega nivoja ustreza zančnemu izrazu, ki nastopa v višjem nivoju kot operator in v katerem vgnezdena zanka zopet nastopa z imenom operatorja, itd.

2. Označitev operatorjev z vrednostjo števca odločitvenih nivojev (ŠON) v vseh izrazih vozlišč drevesne strukture:

ČE (izraz = zančni izraz)

POTEM (velja za izraz znotraj iteracijskega oklepaja)
- a. inicializiraj ŠON na 0
- b. skaniraj izraz od leve proti desni
- c. če detektiraš znak "(" ali "+" ali "-", povečaj ŠON za 1, če pa naletiš na znak ")", zmanjšaj ŠON za 1
- d. ob detekciji operatorja pripiši operatorju trenutno vrednost ŠON

SICER
- a. inicializiraj ŠON na 0
- b. skaniraj izraz od leve proti desni
- c. če detektiraš znak "(" povečaj ŠON za 1, če pa naletiš na znak ")", zmanjšaj ŠON za 1
- d. ob detekciji operatorja pripiši operatorju trenutno vrednost ŠON

3. Uporabi desno distributivno transformacijo (Definicija 3) za vsak kontrolni izraz vozlišča drevesne strukture. Pri tem zaključni operator ($) uporabi kot vse ostale.

4. Za vsako vozlišče drevesne strukture zapiši vi prednostno tabelo PT. Pri vpisovanju vrednosti $o(a_i,a_j)$ v tabelo upoštevaj naslednja:

A. ZA VSAK $o(a_i,a_j)$:

1. ČE $w \vee q$ nastopa v vozliščnem izrazu, kjer sta w,q subizraza, tako, da velja: $a_i \in w$ in $a_j \in q$
   POTEM $o(a_i,a_j) = \not\wedge$

2. ČE ai decizijski operator, aj pa ne ter nj > ni (nj,ni sta številki decizijskega nivoja operatorjev aj, ai)
   POTEM o(ai,aj) = >

3. ČE o(ai,aj) = 0 (še ni specificiran) in aj = $
   POTEM o(ai,aj) = >

4. ČE o(ai,aj) = 0 in $Iai \cap (Vaj \cup Iaj) = 0$ in $Iaj \cap (Vai \cup Iai) = 0$
   POTEM o(ai,aj) = ||
   SICER o(ai,aj) = =>

B. Določi maksimalni niz paralelnih relacij ($\wedge$) v PT z uporabo tranzitivne lastnosti relacije > na naslednji način:

VSAKO vrstico v PT, v smeri od spodaj navzgor, skaniraj od desne proti levi:

1. ČE o(ai,aj) = > in o(aj,ak) = >
   POTEM o(ai,ak) = >
2. ČE o(ai,aj) = ||
   POTEM o(ai,aj) = $\wedge$
3. ČE o(ai,aj) = $\not\wedge$
   POTEM ne spreminjaj vhodov PT

5. Za vsako prednostno tabelo PT iz koraka 4 določi:

A. nize neposredno predhodnih operatorjev, kot sledi:
- izberi kolono po vrsti od desne proti levi
- koloni pripadajočemu operatorju določi niz predhodnih operatorjev. To so operatorji, ki imajo v PT na mestu preseka z opazovano kolono znak > . (Če je operator decizijski, mu dodaj + ali -). Nize označi z NPj, kjer je j oznaka operatorja, kateremu iščemo neposredne predhodnike.
- če je aj,ak NPi in ak NPj, odstrani ak iz NPi.

B. KI dobimo iz NPj na osnovi pravil:

1. ČE NPj = 0
   POTEM KIaj = λ

2. ČE |NPj| = 1 in ai ∈ NPj
   POTEM KIaj = KIai.aj

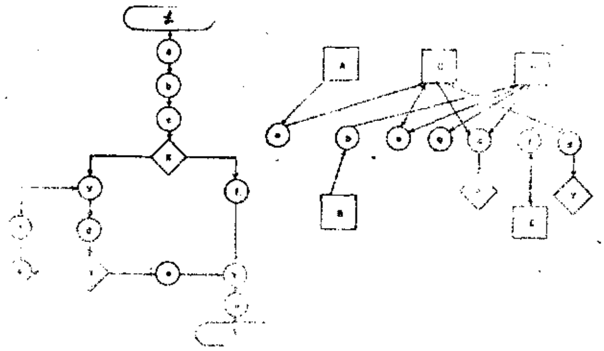
3. ČE |NPj| > 1 in ai ∈ NPj
   POTEM KIaj = $\&_i$KIai.aj

KI = $\bigvee_i$(i) KI[illegible] kjer so $i izhodni operatorji

C. Nariši PG' na osnovi KI

6. V višjih vozliščih [illegible] strukture zamenjaj operatorje za [illegible] nižjih vozlišč.

[illegible] P: Določitev maksimalne stopnje paralelnosti za strukturo [illegible]unanja, ki jo določata $\mathcal{G}$ in $\mathcal{A}$.



Sl. 3. Grafična pred[illegible] 2)

$\mathcal{A}$: KI = a.b.c(- d-e.f .d- [illegible]).g.$ (7)

DIa = ((A),(C))
DIb = ((B),(D))
DIc = ((C,D),(X))
DId = ((C),(Y))
DIe = ((C,D),(C))
DIf = ((E),(E))

Na osnovi Postopka 1 dobimo:

k 1. Drevesna struktura S = (s1,s2)

KI(s1) = a.b.c(-L.e V +f).g.$ (8)
KI(s2) = <d-e.f> [illegible] = L

k 2. KI(s1) = a.b.c.(-L.e V+f).g.$ (9)
(oznake nad operatorji: 0 0 0, 1 1, 1, 0 0)

KI(s2) = <d-e.f> . + (oznake: 0 1 1)

k 3. KI(s1) = a.b.c([illegible] $1 V+f.g2.$2) (10)

KI(s2) = <d-e.f>.d+ (oznake: 0 1 1)

g1 in g2 sta kopiji g, $1 in $2 pa kopiji $.

k 4. PT(s1):

		$2	s2	f	$1	s1	e	L	c	b
		0	0	1	0	0	1	1	0	0
0	a	>	>	>	>	>	>	>	>	∧
0	b	>	>	>	>	>	>	>	>	
0	c	>	>	>	>	>	>	>		
1	L	A	A	A	>	>	>			
1	e	A	A	A	>	>				
0	s1	A	A	A	>					
0	$1	A	A	A						
1	f	>	∧							
0	s2	>								

PT(s2):

		f	e
		1	1
0	d	>	>
1	e	∧	

Sl. 4. Prednostni tabeli za vozlišča drevesa S (Primer 2)

k 5.

A: s1:

j	NPj
a	-
b	-
c	a,b
L	c^-
e	L
g1	e
$1	g1
f	c^+
g2	c^+
$2	g2,f

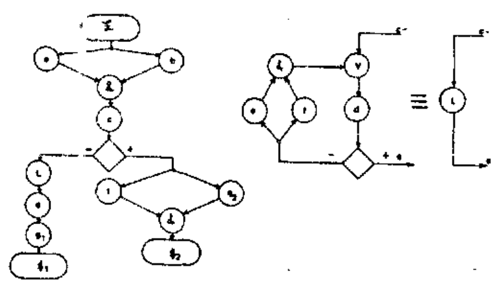
s2:

j	NPj
d	-
e	d^-
f	d^-

B: KI(s1) = KI$1 V KI$2 = (a&b).c-L.e.g1 V c+ +(g2&f) (11)

KI(s2) = c-L = c-<d-(e&f)> .d+

C: PG'(s1): PG'(s2):



Sl. 5. Prednostna grafa za vozlišča drevesa S (Primer 2)

k 6. KI = (a&b).c-<d-(e&f)> .d+e.g1 V c+(g2&f) (12)

IV. Sinteza strukturnih rešitev z VLSI polji na osnovi DF analize

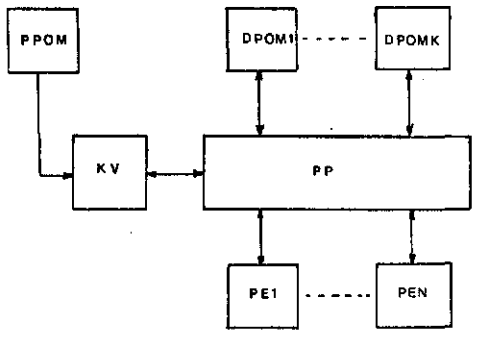
Vzemimo, da smo računanju R, ki ga podaja g oziroma $\mathcal{A}$, na osnovi analize podatkovnih tokov (Poglavje III) določili PG', ki določa maksimalno stopnjo paralelnosti procesiranja N. Tedaj lahko realiziramo dano računanje R s strukturo f, ki vsebuje:

$$f = \langle POM, KV, PV, PP \rangle \quad , \qquad (13)$$

kjer je:

POM = (PPOM, DPOM1, ..., DPOMK)
PPOM - programski pomnilnik
DPOMi - delovni pomnilnik spremenljivke i
KV - krmilno vezje za nadzor procesiranja
PV = (PE1, PE2, ..., PEN)
PEj - procesna enota j
PP - povezovalne poti za prenos podatkov in krmilnih signalov

Splošno blok shemo strukturne rešitve f računanja R podaja slika 6.



Sl. 6. Blok shema strukturne rešitve f računanja R

Značilnost strukture f je v tem, da odpravlja "ozko grlo" z N procesnimi enotami, porazdeljenim delovnim pomnilnikom $DPOM_i$, i=1,...,K in takšnim KV, ki nadzira sočasno delovanje maksimalno N procesnih enot.

Analiza podatkovnih tokov je pokazala, da v vsakem trenutku procesiranja ne potrebujemo vseh PE_j, niti povezav do vseh $DPOM_i$. Zato je dodatna naloga KV, da omogoči le tiste resurse, ki so v danem trenutku potrebni za procesiranje. To pomeni, da morata biti PV in PP izvedena tako, da ju je mogoče spreminjati v realnem času. Slednje predstavlja pomembno zahtevo pri izbiri VLSI komponente. Poleg tega pa je s tem določena tudi vsebina PPOM, ki mora hraniti sekvenco procesiranja ter stanja PV in PP v vsaki fazi procesiranja.

Definicija 4: ULP omogoča homogeno realizacijo strukture f.

Dokaz: Sledi iz Definicije 1 in značilnosti strukture f. Pri tem homogena realizacija pomeni, da so vsi resursi strukture f hkrati zajeti v ULP oziroma regularnem polju ULP.

Zahteva po programiranju funkcij ULP v realnem času zahteva MASTER-SLAVE način vpisovanja lastnosti celic /3/, kar pomeni, da SLAVE register celice hrani podatke o trenutni funkciji celice, MASTER register pa lahko v istem času sprejme podatke o funkciji celice v naslednjem intervalu. Preklop funkcije izvede KV s taktom ure.

Postopek 2: Določitev strukture f računanja R na osnovi analize podatkovnih tokov.

1. Določi prednostne grafe $PG'(s_i)$ za vsa vozlišča drevesne strukture S (Postopek 1).
2. Vsaki izmed PE_j, j=1,...,N, določi niz operatorjev o_i P_j na osnovi minimalnega pokritja /4/.
3. Določi sekvenco računanja ob upoštevanju paralelnosti iz $PG'(s_i)$. Za vsak korak računanja določi aktivne operatorje oz. PE_j, ki jih izvajajo, ter povezave med njimi in $DPOM_i$, ki hranijo vhodno/izhodne spremenljivke aktivnih operatorjev. Slednje določa program procesiranja, ki ga hrani PPOM.
4. Določi KV, ki čita ukaze iz PPOM, aktivira delovne PE_j in ustrezne povezave, ter pripravi podatke za PV in PP v naslednjem koraku.
5. Za vsak blok strukture f določi ustrezno VLSI realizacijo (zavisi od izbire ULP).
6. Homogena realizacija vseh blokov skupaj v okviru enega ULP oziroma polja regularno povezanih ULP, predstavlja strukturno rešitev oziroma arhitekturo danega računanja.

Postopka 1 in 2 skupaj določata prevajalnik od grafične oz. analitične notacije računanja do njegove VLSI realizacije na osnovi ULP. ULP torej realizira poljubne arhitekture, ki jih določamo za vsak primer posebej tako, da upoštevamo vse značilnosti problema s pomočjo analize podatkovnih tokov.

V. Zaključek

V prispevku je podana analiza računanja na osnovi podatkovnih tokov, ki omogoča določitev maksimalne stopnje paralelnosti procesiranja. Na tej osnovi je mogoče zgraditi strukturo, ki je sposobna izvesti dano računanje s takšno stopnjo paralelnosti, kot jo dopušča narava problema. Uporaba ULP ima več prednosti. Omogoča homogeno realizacijo ter dopušča, da s programiranjem sproti vključujemo v procesiranje le aktivne resurse. Takšna izbira pogojuje boljše performančne lastnosti, upošteva pa tudi trende v tehnologiji, kar zagotavlja nizko ceno.

Literatura:

1. A.Dobnikar: Snovanje računalniških struktur z univerzalnim logičnim poljem
9. bosanskohercegovački simpozij iz informatike, Jahorina 85.

2. S.S.Patil, T.A.Welch: A programmable Logic Approach for VLSI
IEEE T.C., Sept. 79.

3. A.Dobnikar, V.Guštin, D.Raič: Interna poročila za nalogo: Snovanje porazdeljenih struktur za procesiranje podatkovnih tokov, RSS, C2-0124, 85.

4. J.Virant: Preklopne strukture in sistemi, Univerza E.Kardelja v Ljubljani, FE, 1983.

5. M.C.Wei, H.A.Sholl: An Expression Model for Extraction and Evaluation of Parallelism in Control Structures
IEEE, T.C., Sept. 82.

THE REST MODULE: COMPUTER SIMULATION OF THE CONTROL UNIT.

G. Aloisio

Dipartimento di Elettrotecnica ed Elettronica
Universita' degli Studi di Bari
Via Re David, 200 - 70125 Bari (Italy)

UDK: 681.3.517.11

ABSTRACT

The REST architecture is based on a bit-slice processor (SLICE) originally designed (1979 - SINBIT project) for "fault-tolerant" structures /1-6/.
The SLICE is a module of low complexity (only three specialized sub-units are present) and of high flexibility (the use of REST modules allows the immediate implementation both of simple structures and of the most complex ones).
More about the REST features is referred in /7/.
A REST simulator to be used both as a preliminary step towards the VLSI implementation of the module and as a hardware "development system" specialized for REST modules, in order to simulate "composite" architectures has been just presented /8/.
The same can work, as evident, as a firmware "development system" for REST based architectures.
A highly parallel architecture based on "REST" type processor elements (RPE) and designed to specialize the REST for the signal processing has been proposed /9/: we used the software simulator for testing this system and convolution and DFT algorithms have been implemented.
The VLSI implementation of a single REST Processing Element (RPE) is also in progress.
The heart of the SLICE is the control unit, containing the firmware for fetching and executing the machine language instructions , the hardware for controlling the sequence of execution of microinstructions and the hardware for the interrupt management. In this paper the computer simulation of the REST control unit is presented.

1. INTRODUCTION

As above mentioned, the SLICE is formed by only three specialized sub-modules (fig.1):

- Controller (C)
- Communication Processor (CP)
- Memory Processor (MP)

In fig.1, IB and XB respectively are the "Internal Bus" (on the slice) and the "External Bus"(outside the slice): the direct communications between the IB and XB buses are maded by the CP sub-module.
It is easy to expand the number of the external bus lines and also the number of buses by a simple replication of this CP sub-module, which is the only sub-module able to process whole words.
The C sub-module can "see" only the whole XB taking from it the "machine instructions".
The MP sub-module operates using only the internal bus IB; direct access to XB are forbidden to it.
A single N-bit processor can be assembled using N different REST modules each of them yielding on an external bus single or multiple bit contribution, but, althogh it is a bit-slice, a single REST-SLICE can work (with a efficiency decrease) as a 16-bit microprocessor, exploiting the inner pre-fetch deepness of the MP sub-module /7/.

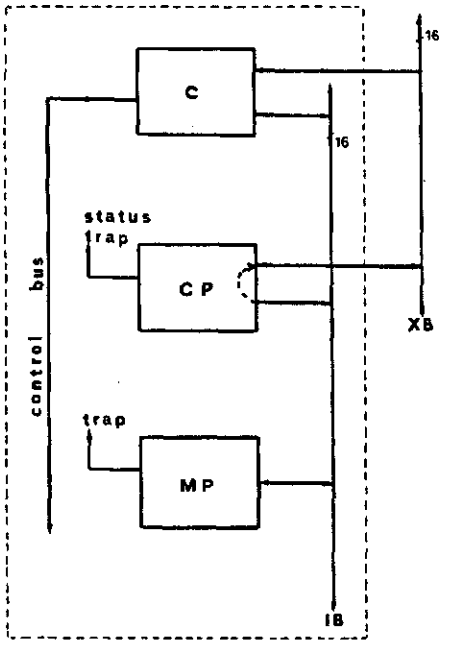


FIG.1 : Simplified REST configuration

2. THE REST CONTROL UNIT

The Slice is a microprogrammed machine, i.e. a coherent sequence of microinstructions is used to execute a machine instruction. Therefore the SLICE control unit consists of the microprogram memory and the structure to determine the address of the next microinstruction.
The block diagram of the REST control unit (C) is shown in fig.2.
The controller can be divided into two principal functional blocks:
- the Sequential Controller (SC), formed by the microprogram memory (MROM) and its control unit (AM2910);
- the Interrupt Controller (IC), for the interrupt detection and management.

2.1 THE SEQUENTIAL CONTROLLER (SC)

The microprogram memory (MROM) is a 4096 word by 76 bit memory and is used to hold the various microinstructions.
For controlling the sequence of execution of microinstructions stored in microprogram memory we used the AMD address sequencer (AM2910): this microprogram controller, besides the capability of sequential addressing, provides conditional branching to any microinstruction within its 4096-microword range.
The AM2910 block diagram is shown in fig.3.

As shown in fig.3, the microprogram controller contains a four-input muliplexer that is used to select either register/counter, direct input, microprogram counter, or stack as the source of the next microinstruction address.
The AM2910 provides 16 functions which select the address of the next microinstruction to be executed. Four of the instructions are unconditional, i.e. their effect depends only on the instruction. Ten of the instructions have an effect which is partially controlled by an external, data-dependent condition. The result of the data-dependent test is applied to CC. If the CC input is low, the test is considered to have passed, and the action specified in the name occurs; otherwise, the test has failed and an alternate (often simply the execution of the next sequential microinstruction) occurs. Three of the instructions have an effect which is partially controlled by the contents of the internal register/counter: unless the counter holds a value of zero , it is held and a different microprogram next address is selected.
The AM2910 instruction set is shown in table 1, while, in fig.4, examples of their execution are reported. The examples given in fig.4 should be interpreted in the following manner: the intent is to show microprogram flow as various microprogram memory words are executed. For example, the "continue" instruction ,(instruction n.14),as shown in fig.4, simply means

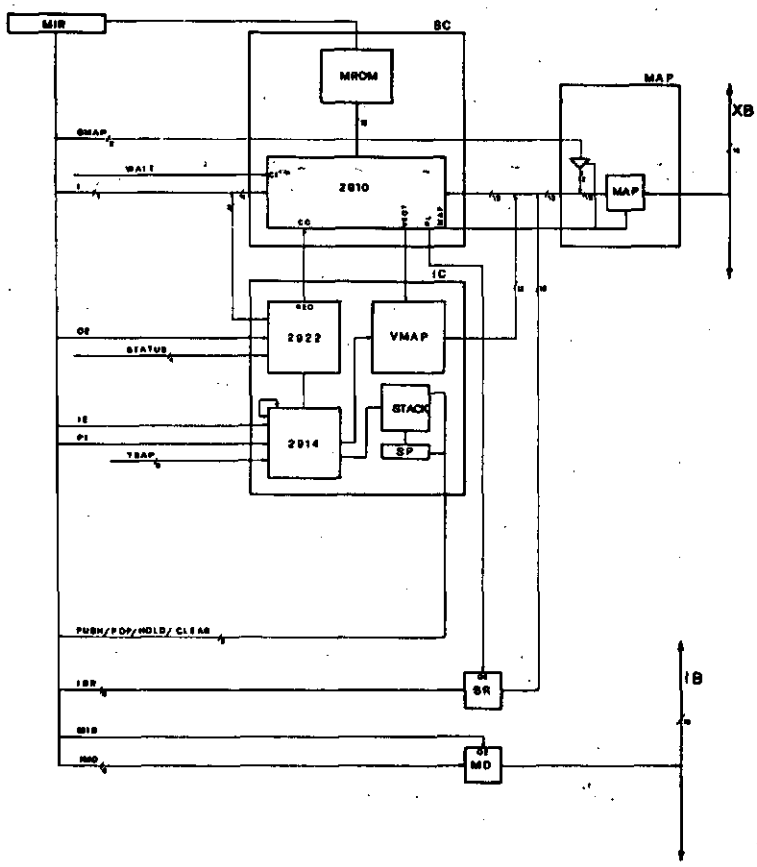


FIG.2 : Block digram of the REST Control Unit

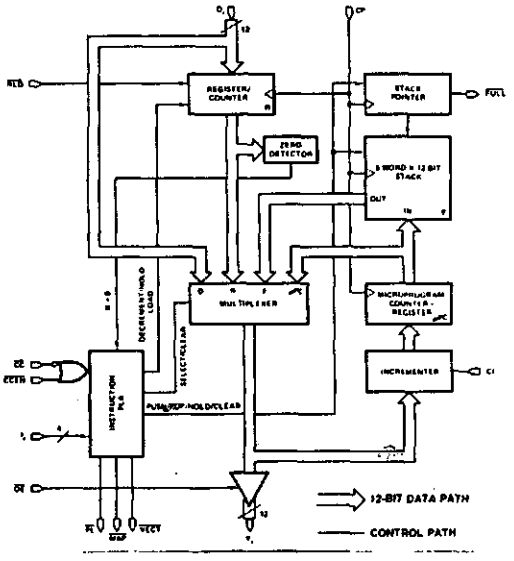


FIG.3 : AM2910 block diagram

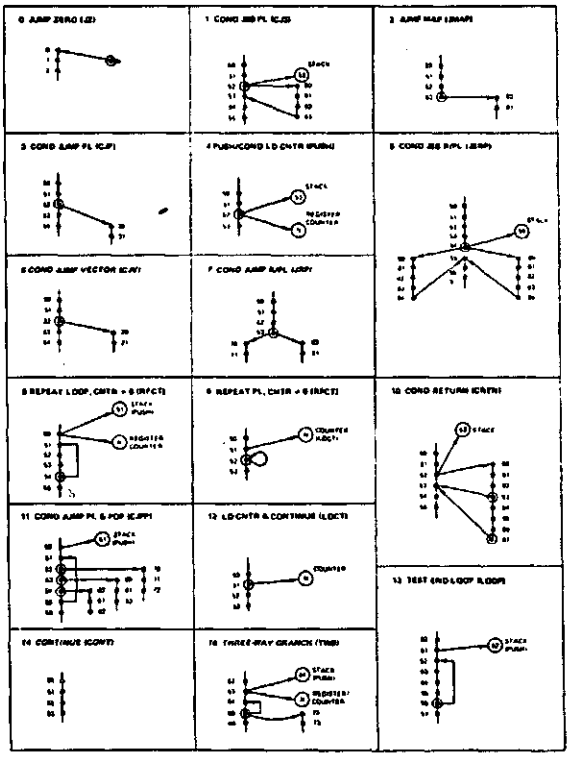


FIG.4 : AM2910 Execution examples

that the contents of microprogram memory words (addresses 50,51,52,53...) are sequentially executed. The microprogram addresse used in the examples were arbitrarily chosen and have no meaning other than to show instruction flow. The exception to this is the first example, "jump zero", which forces the microprogram location counter to address zero. Each dot refers to the time that the contents of the microprogram memory word is in the pipeline register (MIR). When sequencing through continuous microinstructions in microprogram memory, the microprogram counter in the AM2910 is used. Here the control logic simply select the PC input of the multiplexer as the next address. As above mentioned, besides the microprogram counter and the register/counter, other sources of the next microinstruction address are a direct input (D) and the stack (F). Three lines (PL,MAP,VECT) are therefore controlled by the "Instruction PLA" for controlling the three-state outputs of a Branching-Prom (BR), a Mapping-Prom (MAP) and the Interrupt Vector-Prom (VMAP) respectively. The Interrupt Vector-Prom provides one technique for performing interrupt type branching at the microprogram level. The other source of the next microinstruction address (F) is a 5 x 12 stack, used for looping and subrouting in microprogram operations. Up to five levels of subroutines or loops can be nested.

2.2 THE INTERRUPT CONTROLLER

The Interrupt Controller (IC) allows the detection and the handling of asincronous requests (interrupts) that happen within the SLICE.
The AM2914 is the heart of the Interrupt Control Unit: it can be microprogrammed to meet the specific application's requirement. The AM2914 block diagram and the logic symbol are shown in fig.5 and fig.6 respectively. Sixteen different microinstructions are possible, controlled by a four-bit microinstruction field (PI). The AM2914 microinstruction set is shown in fig.7.
As shown in fig.5, a "microinstruction decode" circuitry decode , if there is the "Instruction Enable",these Interrupt Microinstructions and generates required control signals for the chip.

HEX I3I0	MNEMONIC	NAME	REG/ CNTR CONTENTS	FAIL $\overline{CCEN}$ = LOW and $\overline{CC}$ = HIGH Y	FAIL STACK	PASS $\overline{CCEN}$ = HIGH or $\overline{CC}$ = LOW Y	PASS STACK	REG/ CNTR	ENABLE
0	JZ	JUMP ZERO	X	0	CLEAR	0	CLEAR	HOLD	PL
1	CJS	COND JSB PL	X	PC	HOLD	D	PUSH	HOLD	PL
2	JMAP	JUMP MAP	X	D	HOLD	D	HOLD	HOLD	MAP
3	CJP	COND JUMP PL	X	PC	HOLD	D	HOLD	HOLD	PL
4	PUSH	PUSH/COND LD CNTR	X	PC	PUSH	PC	PUSH	Note 1	PL
5	JSRP	COND JSB R/PL	X	R	PUSH	D	PUSH	HOLD	PL
6	CJV	COND JUMP VECTOR	X	PC	HOLD	D	HOLD	HOLD	VECT
7	JRP	COND JUMP R/PL	X	R	HOLD	D	HOLD	HOLD	PL
8	RFCT	REPEAT LOOP CNTR ≠ 0	≠ 0	F	HOLD	F	HOLD	DEC	PL
			= 0	PC	POP	PC	POP	HOLD	PL
9	RPCT	REPEAT PL,CNTR ≠ 0	≠ 0	D	HOLD	D	HOLD	DEC	PL
			= 0	PC	HOLD	PC	HOLD	HOLD	PL
A	CRTN	COND RTN	X	PC	HOLD	F	POP	HOLD	PL
B	CJPP	COND JUMP PL & POP	X	PC	HOLD	D	POP	HOLD	PL
C	LDCT	LD CNTR & CONTINUE	X	PC	HOLD	PC	HOLD	LOAD	PL
D	LOOP	TEST END LOOP	X	F	HOLD	PC	POP	HOLD	PL
E	CONT	CONTINUE	X	PC	HOLD	PC	HOLD	HOLD	PL
F	TWB	THREE WAY BRANCH	≠ 0	F	HOLD	PC	POP	DEC	PL
			= 0	D	POP	PC	POP	HOLD	PL

Note 1: $\overline{CCEN}$ = LOW and $\overline{CC}$ = HIGH, hold, else load X = Don't Care

TABLE 1 : AM2910 Microinstruction set

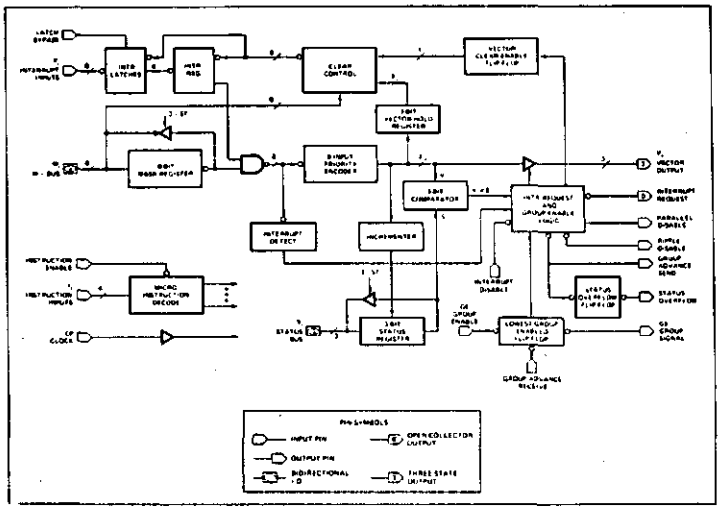


FIG.5 : AM2914 Block diagram

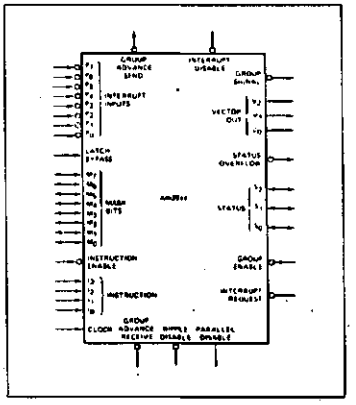


FIG.6 : AM2914 Logic Symbol

The AM2914 receives interrupt requests (TRAP) on 8 interrupt input lines which are held by an eight-bit "interrupt register". These requests are ANDed with the corresponding bit in the "mask-register" and the results are sent to an 8-input "priority encoder", which produces a three bit encoded vector representing the highest numbered input which is not masked.
An internal 3-bit "status register" is used to point to the lowest priority at which an interrupt will be accepted. The contents of the status register are compared with the output of the priority encoder, and an interrupt request output will occurr if the vector is greater than or equal to status. Whenever a vector is read from the AM914, the status register is automatically updated to point to one level higher than the vector read. An interrupt request and a 3-bit "vector out" are then produced by the AM2914: the interrupt request is applied through a multiplexer (AM2922) to the "Condition Code" input (CC) of the AM2910, while the vector out, by means a "Vector Mapping" Prom (VMAP), provides a substitute microprogram address corresponding to the location of the "interrupt service routine".

MICROINSTRUCTION DESCRIPTION	MICROINSTRUCTION CODE $i_3i_2i_1i_0$
MASTER CLEAR	0000
CLEAR ALL INTERRUPTS	0001
CLEAR INTERRUPTS FROM M-BUS	0010
CLEAR INTERRUPTS FROM MASK REGISTER	0011
CLEAR INTERRUPT, LAST VECTOR READ	0100
READ VECTOR	0101
READ STATUS REGISTER	0110
READ MASK REGISTER	0111
SET MASK REGISTER	1000
LOAD STATUS REGISTER	1001
BIT CLEAR MASK REGISTER	1010
BIT SET MASK REGISTER	1011
CLEAR MASK REGISTER	1100
DISABLE INTERRUPT REQUEST	1101
LOAD MASK REGISTER	1110
ENABLE INTERRUPT REQUEST	1111

FIG.7 : AM2914 Microinstruction set

3. THE REST CONTROL UNIT SIMULATION

The hardware/firmware development system we implemented and specialized for REST modules is written in Pascal.
The Pascal peculiarity to be a structured language can be efficiently exploited for a structured architectures descriptions.
In our simulation case, a hardware realization of the REST controller was equivalent to the software implementation of a Pascal procedure, that exactly simulate the behaviour of the Sequential Controller (SC) and the Interrupt Controller (IC) and that have been tested separately according to their functional characteristics.
For the simulation of memory elements, presented on the controller (HROM, VMAP, MAP, BR, MD), we used normal data files of suitable dimensions : interactive tasks for the data read/write operations are provided. An interactive system for writing of microprograms has been also realized.
The flow-chart of the Rest Controller simulation program is shown in fig.8.
INITP is a normal initialization procedure, used essentially for resetting operations ; the microprogram lenght and the inital value of the internal bus (IB) and the external bus (XB) are also assigned in this phase.
The asincrounous interrupts generation is made by means the "INT" procedure.

The interrupt generation occurs before the execution of every microinstruction in order to simulate the AM2910 conditional instructions: as above mentioned, the AM2914 output -IR- (i.e. the interrupt request) is multiplexed with the "status-word" (STATUS) by the AM2922 and used for controlling the AM2910 "condition input" (CC).
PRC is the simulation procedure of the machine program-counter.
The state of the internal and the external buses is also reported for every firmware step.
All the 16 AM2910 instructions (see fig.4) have been simulated and tested: the output of the simulation program for these examples and the relative firmware are reported in fig.9 and in fig.10 respectively.
For lack of space, only a part of these outputs in simplified form is provided. The output of the simulation program should be interpreted in the following manner: in the first interactive phase the start value of the XB bus is requested and we use this value to communicate the starting address of the testing microprogram. Then after the interrupt generation and an initial "reset phase" the simulation program of the Controller can start: the simulation procedures of the controller are sequentially activated, according to the microinstruction executed.
In the "reset phase" the location "zero" of the microprogram memory (MROM) is forced, i.e. a "Jump-zero" instruction is simulated. As shown in fig.10, in the location "zero" of MROM the AM2910 "Jump Map" instruction is written, so that the XB value is used to calculate the first address of the microprogram.
The microprogram of fig.10, can be used for testing some AM2910 instruction (see fig.4), such as the "Conditional Jump to Subroutine" (inst.n.1 - CJS), the "Repeat loop counter" (inst.n.8 - RFCT), the "Conditional Jump to Vector" (inst.n.6 - CJV) etc.
The microprogram flow as various microprogram memory words are executed can be followed looking over the address generated by the AM2910 subprocedure (YO).

CONCLUSIONS

The two principal blocks of the REST controller, the Sequential Controller (SC) used for controlling the sequence of execution of microinstruction stored in the microprogram memory, and the Interrupt Controller (IC), used for the interrupt detection and management have been described: the behaviour of the controller has been tested by means of a software simulator written in Pascal.
Firmware examples and the relative simulation program outputs for the controller functionality simulation and testing are also reported.

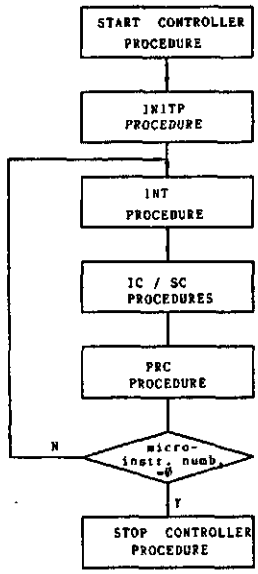


FIG.8 : REST Controller Simulation flow chart

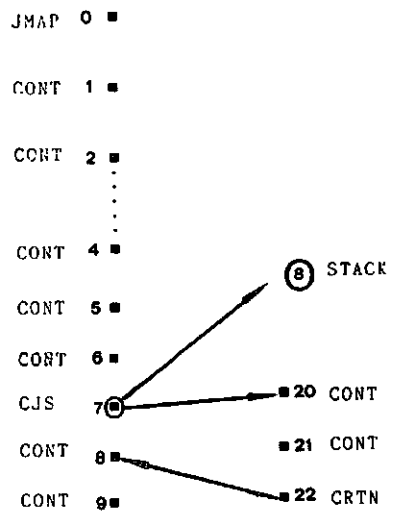
REFERENCES

/1/ D.Marino: "Progetto SINBIT, 1979." Report CNR (P.F.I.) 79-P1-Sinbit-CNR. Roma-Italy
/2/ M.De Blasi, G.Degli Antoni, M.Mallamo, D.Marino: "Progetto di calcolatore Fault-tolerant Single-Processor ad elevato grado di emulazione."Proc.AICA 1979, Bari-Italy.
/3/ M.De Blasi, G.Degli Antoni, D.Marino: "Proposta di architettura multiprocessor con capacita' di riconfigurazione a piu' livelli." Proc.AICA, Oct 1979. Bari-Italy.
/4/ M.De Blasi, D.Marino: "The SINBIT reconfigurable computer: Architectural insights." Proc.AICA, Oct 1980.Bologna-Italy
/5/ G.Aloisio, P.Dello Russo, M.Mallamo: " Calcolatore SINBIT: unita' componenti dello slice" Proc.AICA. Oct.1980. Bologna-Italy
/6/ G.Aloisio, F.Corvino, M.Mallamo, D.Marino: "Memory management and protection methods in the SINBIT experimental computers" 17 Conference on Microcomputer Tecnology and Usage- Ljubljana June 1983.
/7/ D.Marino: "A powerful element for reconfigurable processing structure" 2 Conference on Image Analysis and Processing. 1982. Fasano -Italy.
/8/ G.Aloisio: "A computer simulation of a single-bit REST module" ISMM International Journal of Mini and Microcomputers, in press.
/9/ G.Aloisio, V. Di Lecce: " A REST module based multiprocessor system for signal processing" 29 International Symposium "Mini and Microcomputers and Their Applications" - Sant Feliu de Guixols - Spain (June 25-28, 1985)
/10/ G.Aloisio, N.Veneziani: "A proposal for the implementation of a BAP image processor using REST modules"
24 International Symposium "Mini and Microcomputers" Bari-Giugno 1984

```
>RUN IESICO
MICROINSTRUCTIONS NUMBER ? = 10
WAIT ? (Y/N)N
##############################
                    M..:...:...:...L
START VALUE OF XB  = 0000000000000001
START VALUE OF IB  = 0000000000000000

##############################
        INTERRUPT GENERATION
##############################
          16.............1
STATUS = 0000000000000000
          16.............1
TRAP   = 0000000000000000
##############################

START RESET (JUMP ZERO)

##############################
START CONTROLLER-PROCEDURE

                SC-subprocedure
                AM2910-subprocedure
AM2910 INSTR. = JUMP ZERO
Y0 GENERATED BY AM2910 =      0
                DMIR-subprocedure
XB VALUE  = 0000000000000001
IB VALUE  = 0000000000000000
STOP FETCH OF MICROINSTRUCTION N.      1

##############################
        INTERRUPT GENERATION
##############################
          16.............1
STATUS = 0000000000000000
          16.............1
TRAP   = 0000000000000000
##############################

START EXECUTION OF MICROINSTRUCTION N.      1

##############################
START CONTROLLER-PROCEDURE

                SC-subprocedure
                AM2910-subprocedure
AM2910 INSTR. = JUMP MAP
                MAP-subprocedure
Y0 GENERATED BY AM2910 =      4
                DMIR-subprocedure
XB VALUE  = 0000000000000001
IB VALUE  = 0000000000000000
STOP FETCH OF MICROINSTRUCTION N.      2

##############################
        INTERRUPT GENERATION
##############################
                    .
                    .
                    .
                    .
                    .

START EXECUTION OF MICROINSTRUCTION N.      4

##############################
START CONTROLLER-PROCEDURE

                SC-subprocedure
                AM2910-subprocedure
AM2910 INSTR. = CONTINUE
Y0 GENERATED BY AM2910 =      7
                DMIR-subprocedure
XB VALUE  = 0000000000000001
IB VALUE  = 0000000000000000
STOP FETCH OF MICROINSTRUCTION N.      5

##############################
        INTERRUPT GENERATION
##############################
          16.............1
STATUS = 0000000000000001
          16.............1
TRAP   = 0000000000000000
##############################

START EXECUTION OF MICROINSTRUCTION N.      5

##############################
START CONTROLLER-PROCEDURE

                SC-subprocedure
                AM2910-subprocedure
AM2910 INSTR. = CONDITIONAL JSB PL
                BR-subprocedure
                IC-subprocedure
                AM2922-subprocedure
Y0 GENERATED BY AM2910 =      20
                DMIR-subprocedure
XB VALUE  = 0000000000000001
IB VALUE  = 0000000000000000
STOP FETCH OF MICROINSTRUCTION N.      6

                    .
                    .
                    .
                    .
                    .

START EXECUTION OF MICROINSTRUCTION N.      7

##############################
START CONTROLLER-PROCEDURE

                SC-subprocedure
                AM2910-subprocedure
AM2910 INSTR. = CONTINUE
Y0 GENERATED BY AM2910 =      22
                DMIR-subprocedure
XB VALUE  = 0000000000000001
IB VALUE  = 0000000000000000
STOP FETCH OF MICROINSTRUCTION N.      8

##############################
        INTERRUPT GENERATION
##############################
          16.............1
STATUS = 0000000000000001
          16.............1
TRAP   = 0000000000000000
##############################

##############################

START EXECUTION OF MICROINSTRUCTION N.      8

##############################
START CONTROLLER-PROCEDURE

                SC-subprocedure
                AM2910-subprocedure
AM2910 INSTR. CONDITIONAL RETURN
                BR-subprocedure
                IC-subprocedure
                AM2922-subprocedure
Y0 GENERATED BY AM2910 =      8
                DMIR-subprocedure
XB VALUE  = 0000000000000001
IB VALUE  = 0000000000000000
STOP FETCH OF MICROINSTRUCTION N.      9

##############################
        INTERRUPT GENERATION
##############################
          16.............1
STATUS = 0000000000000000
          16.............1
TRAP   = 0000000000000000
##############################

START EXECUTION OF MICROINSTRUCTION N.      9

##############################
START CONTROLLER-PROCEDURE

                SC-subprocedure
                AM2910-subprocedure
AM2910 INSTR. = CONTINUE
Y0 GENERATED BY AM2910 =      9
                DMIR-subprocedure
XB VALUE  = 0000000000000001
IB VALUE  = 0000000000000000
STOP FETCH OF MICROINSTRUCTION N.      10
##############################
     STOP CONTROLLER PROCEDURE
##############################
EXECUTION TIME OF MICROPROGRAM =   3000 ns
>_
```

FIG.9 : Example of simulation program output for the AM2910 "Jump to Subroutine" instruction

```
**********************************************************************
                    MROM-LOCATION  N.      0
SMAP   I   I22 OE IE  PI  STC   IBR    MIB   IMD      FAU      NAD   LNR
 00  0010 000  0  0 0000 00  00000000  0  00000000 00000000 00000000  0

  LRE  LWH  RAU  RNR  RWR  TIX  R  L  RA LA   FF     NA      LN RN  TXI
   0    0    0    0    0    0   0  0  0  0   0000  00000000  0  0   0
**********************************************************************
                                  .
                                  .
                                  .
                                  .
                                  .
**********************************************************************
                    MROM-LOCATION  N.      4
SMAP   I   I22 OE IE  PI  STC   IBR    MIB   IMD      FAU      NAD   LNR
 00  1110 000  0  0 0000 00  00000000  0  00000000 00000000 00000000  0

  LRE  LWH  RAU  RNR  RWR  TIX  R  L  RA LA   FF     NA      LN RN  TXI
   0    0    0    0    0    0   0  0  0  0   0000  00000000  0  0   0
**********************************************************************

**********************************************************************
                    MROM-LOCATION  N.      5
SMAP   I   I22 OE IE  PI  STC   IBR    MIB   IMD      FAU      NAD   LNR
 00  1110 000  0  0 0000 00  00000000  0  00000000 00000000 00000000  0

  LRE  LWH  RAU  RNR  RWR  TIX  R  L  RA LA   FF     NA      LN RN  TXI
   0    0    0    0    0    0   0  0  0  0   0000  00000000  0  0   0
**********************************************************************
                                  .
                                  .
                                  .
                                  .
**********************************************************************
                    MROM-LOCATION  N.      7
SMAP   I   I22 OE IE  PI  STC   IBR    MIB   IMD      FAU      NAD   LNR
 00  0001 001  1  0 0000 00  00000000  0  00000000 00000000 00000000  0

  LRE  LWH  RAU  RNR  RWR  TIX  R  L  RA LA   FF     NA      LN RN  TXI
   0    0    0    0    0    0   0  0  0  0   0000  00000000  0  0   0
**********************************************************************

**********************************************************************
                    MROM-LOCATION  N.      8
SMAP   I   I22 OE IE  PI  STC   IBR    MIB   IMD      FAU      NAD   LNR
 00  1110 000  0  0 0000 00  00000000  0  00000000 00000000 00000000  0

  LRE  LWH  RAU  RNR  RWR  TIX  R  L  RA LA   FF     NA      LN RN  TXI
   0    0    0    0    0    0   0  0  0  0   0000  00000000  0  0   0
**********************************************************************

**********************************************************************
                    MROM-LOCATION  N.      9
SMAP   I   I22 OE IE  PI  STC   IBR    MIB   IMD      FAU      NAD   LNR
 00  1110 000  0  0 0000 00  00000000  0  00000000 00000000 00000000  0

  LRE  LWH  RAU  RNR  RWR  TIX  R  L  RA LA   FF     NA      LN RN  TXI
   0    0    0    0    0    0   0  0  0  0   0000  00000000  0  0   0
**********************************************************************
                                  .
                                  .
                                  .
                                  .
                                  .
                                  .
                                  .
                                  .
                                  .
**********************************************************************
                    MROM-LOCATION  N.     20
SMAP   I   I22 OE IE  PI  STC   IBR    MIB   IMD      FAU      NAD   LNR
 00  1110 000  0  0 0000 00  00000000  0  00000000 00000000 00000000  0

  LRE  LWH  RAU  RNR  RWR  TIX  R  L  RA LA   FF     NA      LN RN  TXI
   0    0    0    0    0    0   0  0  0  0   0000  00000000  0  0   0
**********************************************************************

**********************************************************************
                    MROM-LOCATION  N.     21
SMAP   I   I22 OE IE  PI  STC   IBR    MIB   IMD      FAU      NAD   LNR
 00  1110 000  0  0 0000 00  00000000  0  00000000 00000000 00000000  0

  LRE  LWH  RAU  RNR  RWR  TIX  R  L  RA LA   FF     NA      LN RN  TXI
   0    0    0    0    0    0   0  0  0  0   0000  00000000  0  0   0
**********************************************************************

**********************************************************************
                    MROM-LOCATION  N.     22
SMAP   I   I22 OE IE  PI  STC   IBR    MIB   IMD      FAU      NAD   LNR
 00  1010 001  1  0 0000 00  00000000  0  00000000 00000000 00000000  0

  LRE  LWH  RAU  RNR  RWR  TIX  R  L  RA LA   FF     NA      LN RN  TXI
   0    0    0    0    0    0   0  0  0  0   0000  00000000  0  0   0
**********************************************************************
```



I = 2 (JMAP)
I = 14 (CONT)
I = 1 (CJS)
I = 10 (CRTN)

FIG. 10 : Firmware for the simulation of the AM2910 "Jump to Subroutine" instruction

PRILAGOĐAVANJE RAČUNARSKOG SISTEMA PROŠIRENJU TERMINALSKE MREŽE

Zoran Lazarov, mr ing. , GRO "Pelagonija",
Skopje, Jugoslavija

UDK: 681.3.007

U radu su opisane aktivnosti koje se trebaju preduzeti da bi se omogućilo dvostruko povećanje broja terminala koji bi i dalje bili podržani postojećim resursima. Računarski sistem kojim se rad bavi je VAX-11/780. Najpre se određuje reprezentativno radno opterećenje sistema, zatim se snima postojeće stanje na sistemu merenjem parametara učinka, pa se na osnovu očekivanog proširenja i planova razvoja pretpostavlja radno opterećenje u novim uslovima i vrši njegova simulacija korišćenjem postojećih i sintetičkih poslova. Merenjem novih parametara učinka uočena je njihova degradacija i preduzete su mere radi poboljšanja rada sistema. Izvršena je promena nekih od parametara generisanja, i predložen je postupak za prethodno raspoređivanje poslova prema njihovim zahtevima za procesorom i memorijom. To je rezultovalo značajnim poboljšanjem vrednosti parametara učinka sistema.

TUNING OF A COMPUTER SYSTEM TO THE TERMINAL NETWORK ENLARGEMENT - The paper describes the activities required to enable connecting of twice the previous number of terminals supported by the existing resources. The computer system considered is VAX-11/780. First the representative workload of the system is determined, then the various performance evaluation parameters are measured to record the current state of the system. The new workload for a system with doubled number of terminals is created out of the existing jobs and new synthetic jobs, as well. In order to improve the system performance parameters some of the system generation parameters are changed, and a prescheduling algorithm is proposed that discriminates the jobs according to their processor and memory requirements. Finally, much better values for the performance evaluation parameters are obtained resulting thus in a better tuned system.

Zadatak autora ovog rada je bio: bez nabavljanja dopunske opreme omogućiti povezivan je dvostruko većeg broja terminala od postojećeg. Znači, samo sa preraspodelom i prilagođavanjem postojećih resursa potrebno je ostvariti zadovoljavajuće opsluživanje udvojenog broja korisnika. Računarski sistem kojim se rad bavi je VAX-11/780 sa 768 KB memorije, 3x 67 MB prostora na diskovima, i pored ostalog, jednim asinhronim multipleksorom za 8 terminala. Proširenje se sastoji u nabavljanju još jednog multipleksora sa još 8 terminala.

Radi dobijanja prave slike stanja sistema potrebno je izvršiti snimanje i analizu aktuelnog radnog opterećenja. Time bi se odredili najvažniji korisnici i njihove potrebe, najzastupljeniji poslovi i njihova učestanost, od kojih se zatim može načiniti reprezentativni uzorak opterećenja prisutnog na sistemu. Zatim ostaje da se izvrši merenje parametara učinka sistema u uslovima definisanim reprezentativnim radnim opterećenjem. Rezultati tih merenja t.j. dobijeni pokazatelji učinka se upoređuju sa onima koji postoje iz prethodnog perioda praćenja učinka sistema, čime se u stvari vrši verifikacija reprezentativnosti odabranog radnog opterećenja. Parametri koji su bili mereni jer se smatra da se njima najbolje opisuje rad nekog korisnika su:

- vreme provedeno na sistemu,
- utrošeno procesorsko vreme,
- broj U/I operacija na diskovima,
- broj U/I operacija na terminalu i štampaču,
- broj donošenja stranica u radni komplet procesa.

Dobro slaganje pokazatelja učinka je bilo garancija da će i novopretpostavljeno radno opterećenje kojim se trebaju simulirati uslovi na sistemu sa promenjenom konfiguracijom biti realni odraz ponašanja stvarnog sistema.

U suštini se radi o hibridnom pristupu formiranju radnog opterećenja. Gdegod je to bilo moguće, u reprezentativni niz su bili uključeni stvarni poslovi u svom punom obimu. Međutim, onamo gde je to bilo neizvodljivo, bilo zbog karaktera posla - na primer terminalska sesija, ili zbog nekih njegovig karakteristika - na primer suviše dugačak proces, takvi predstavnici su bili zamenjeni sintetičkim poslovima koji su uglavnom sadržali sve bitne odlike originalnih procesa. Treba naglasiti da su periodi merenja parametara učinka sistema u slučaju reprezentativnih radnih opterećenja uvek kraći od jednog sata, jer je to vreme da bi se izvršila merenja i uočili trendovi dovoljno, a ipak podnošljivo dugo traje kao period u kome je zbog simulacije i merenja pristup sistemu uskraćen svim ostalim korisnicima. Proces kojim je bila simulirana tipična terminalska sesija bio je sastavljen od više poziva raznih korisničkih i sistemskih programa pomoću kojih se manipuliše datotekama, kao što je na primer EDITOR, a koji su svi međusobno povezani komandama DCL (komandni jezik operativnog sistema VAX/VMS). Tokom egzekucije čitavog mernog paketa kojim se simuliralo radno opterećenje stvarnog sistema, vršena su prethodno pomenuta merenja, a pored njih mereno je i vreme odziva kod najčešće korištenih komandi i poziva sistemskim ili korisničkim programima. To vreme upoređeno sa vremenom zadovoljavanja tih zahteva na "praznom" sistemu (neopterećenim nikakvim drugim poslom) daje takozvani faktor produženja - "stretch factor", koji je možda najbolji pokazatelj kvaliteta usluga koje korisnici dobijaju od sistema.

MODELIRANJE I PRILAGOĐAVANJE

Pošto se je na opisani način utvrdila metodologija rada, moglo se je zatim ići na postupak određivanja parametara sistema u hipotetičnim uslovima dvostruko većeg broja terminala. Korištenjem mogućnosti koje pruža VAX VMS (sistem servisi) moguće je kreiranje odvojenih (detached) procesa kojima se mogu dati različita obeležja, već prema potrebi: od simulacije interaktivne terminalske sesije do najjednostavnije serijske (batch) obrade istestiranim programima. Anketiranjem korisnika i razmatranjem razvojnih planova organizacije dobijena su saznanja pomoću kojih je određen obim posla koji se sa prilično sigurnošću može očekivati na novokonfiguriranom sistemu. Zatim je oblikovan reprezentativni uzorak za odgovarajuće očekivano radno opređenje koji je kao i u prethodnom slučaju bio hibridnog karaktera, sa stvarnim i sintetiziranim poslovima. Reakcija sistema, t.j. njegovi parametri učinka su bili prilično loši što je i bilo očekivano s obzirom da nije bilo nikakve akcije prilagođavanja. Budući da su svi procesi imali isti bazni prioritet i podjednaku veličinu radnog kompleta, a zbog male raspoložive memorije po procesu, došlo je do velikog povećanja aktivnosti straničenja i donošenja i odnošenja čitavih procesa u i iz memorije. Rezultat toga bio izuzetno velik broj U/I operacija na diskovima, a uz to, ili bolje, zbog toga, vreme odaziva je bilo uglavnom veoma produženo. Koraci koji su bili preduzeti su bili sledeći: određivanje karaktera pojedinih procesa u radnom opterećenju, utvrđivanje njihovih memorijskih zahteva, podešavanje veličine njihovih radnih kompleta u skladu s tim, podešavanje baznih prioriteta sa favorizovanjem interaktivnih sesija radi održavanja principa pružanja najbržeg odziva onamo gde se najviše očekuje. Zajedno sa tim izvršena je promena nekih od sistemskih parametara generisanja, kao što su parametri za automatsko podešavanje veličine radnog kompleta, za broj pomoćnih paketa za U/I zahteve, kvantum vremena provedenog u memoriji i sl.

POSTUPAK ZA RASPOREĐIVANJE

Da bi tako izvedeno prilagođavanje rezultiralo još boljim pokazateljima učinka sistema bilo je potrebno izvršiti izvesno prerasporedi vanje poslova koji sačinjavaju radno opterećenje. To je učinjeno drugačijom organizacijom posla na sistemu. Naime, svi mogući poslovi su svrstani u tri različite kategorije prema tipu opterećenja koje unose u sistem. U prvoj su bili pretežno interaktivni procesi sa malim zahtevima za procesorskim vremenom, ali sa druge strane veoma osetljivi na dužinu vremena odziva. Drugu su činili poslovi razvijanja i testiranja programa, t.j. mešoviti interaktivni poslovi sa mnogo većom, ali ipak ograničenom potrebom za procesorskim vremenom. Na kraju, u treću grupu su spadali, uslovno rečeno, tipično eksploataciono orijentisani poslovi sa velikim obradama velikog broja podataka i mnogo utrošenog procesorskog vremena. Takva raspodela uslovila je i organizaciju posla na sistemu. Svi procesi iz prve grupe se raspoređuju sa najvišim prioritetom ali sa veoma ograničenim procesorskim vremenom, zatim dolaze procesi sa nižim prioritetom ali dužim iznosom procesorskog vremena, i na kraju procesi koji nemaju vremenskog ograničenja ali im je prioritet najniži. Ova raspodela se temelji na rezonovanju da će procesi sa najvišim prioritetom brzo dobiti procesor, ali će go zbog svog karaktera (pretežno U/I) i brzo osloboditi, dobijajući na taj način svoj neophodni brzi odziv, i ne izazivajući pritom zastoj kod ostalih procesa. Korisnici su motivisani za poštovanje discipline raspoređivanja zbog nemogućnosti obavljanja poslova ukoliko se prekorači određeno vreme, a između serijskih repova iz kojih se inicira i ograničava egzekucija procesa, u grupama sa nižim prioritetima postoji povratna veza koja omogućava prelazak poslova (koji čekaju) iz jednog u drugi rep, kao i dinamično podešavanje prioriteta ukoliko neki od praćenih pokazatelja učinka indicira da je opterećenje na sistemu takvo da se to može dopustiti.

NEKOLIKO REZULTATA I ZAKLJUČNA RAZMATRANJA

Upoređujući novodobivene rezultate sa onima iz ranijih merenja na "praznom" sistemu, kao i na sistemu sa 8 korisnika, dobija se predstava koliko iznosi promena učinka prouzrokovana promenom radnog opterećenja. Najindikativnije u tu svrhu može poslužiti vreme odziva odnosno njegov faktor produženja. U tablici su navedeni faktori produženja glavnih tipova procesa od kojih je bilo sastavljeno radno opterećenje (respektivno, terminalska sesija, razvijanje programa, analiza konstrukcija, lični dohodak) za sisteme sa 8 i 16 korisnika, i odgovarajuća apsolutna vremena na praznom sistemu:

Proces	Vreme na "praznom" sistemu (min.)	Faktor produženja 8 koris.	16 koris.
TERM	8:11	2,7	4,3
CMLK	6:04	2,8	4,8
RZKOS	5:53	2,5	5,2
PLATA	9:08	2,7	5,7

Kao što se i moglo očekivati, najmanji faktor produženja imaju procesi koji spadaju u takozvanu prednju obradu, za razliku od onih iz pozadinske, što je i normalno, sa obzirom na njihov koncipirani položaj prilikom raspoređivanja za obradu.

U sledećoj tablici su navedeni faktori produženja za nekoliko od najčešće korištenih interaktivnih komandi:

Komanda	Vreme na "praznom" sistemu (sek.)	Faktor produženja 8 koris.	16 koris.
DIRECTORY	15	4,1	4,3
COPY (1 disk)	2	3	5,4
COPY (2 diska)	2,2	3,2	6
DEL (1ooo bl.)	1	2,5	6
DEL (2oo bl.)	1	2	6
EDI/FIND	2	1,6	2,2
EDI/INSERT	1	1,4	1,8
EDI/DELETE	1	1,4	1,9
EDI/SUBST	1	1,4	2

Ukupni faktor produženja za sve komande iz skupa komandi iznosi 4 nasuprot vrednosti od 2,8 za isti skup komandi na sistemu sa 8 korisnika.

Ono što je bilo rečeno prilikom merenja u prethodnim izlaganjima važi i ovde. Vreme odaziva je komponenta koja trpi najviše promena, dok se druge manje ili veoma malo menjaju. Još više, vreme odziva je glavni, a često i jedini kriterijum kojim korisnici vrednuju sistem i usluge koje im on pruža. Zbog toga je kod svih merenja i prezentiranja rezultata njemu bilo posvećeno najviše pažnje.

U posljednjoj tablici su prikazani neki od parametara učinka sistema za 16 i (u zagradi) 8 korisnika, koji su dobiveni DISPLAY programom (svedeni na broj u sekundi).

direktne U/I operacije	48,04	(40,24)
spremničke U/I operacije ..	12,42	(10,45)
donošenja stranica	42,27	(35,61)
učitane stranice	18,83	(11,12)
zapisane stranice	14,38	(5,75)
otvorene datoteke	1,92	(1,21)
unošenja	0,91	(0,01)
donošenja za oper. sistem .	1,47	(1,22)

Konačno merenje pokazatelja učinka je potvrdilo da se sistem, doduše veoma opterećen, ponaša u skladu sa očekivanjima i dobro podnosi radno opterećenje. Faktor produženja se kreće oko 4 i 5 što je, s obzirom na uslove, sasvim u okviru optimističkih prognoza. Što je još važnije, analiza kumulativnih pokazatelja učinka sistema je pokazala i potvrdila postojanje visokog stepena prilagođenosti, jer skori i nije bilo, ili je to bilo veoma kratko, procesa spremnih za obradu koji su bili izvan memorije (computable outswapped) što je glavni pokazatelj "rasštimovanosti" sistema.

Treba dodati da je na sistemu ostala otvorena mogućnost takozvanog "finog" prilagođavanja, ali ona u sadašnjem stanju nije razmatrana zbog nestazmernosti odnosa poboljšanja koje bi se moglo postići sa troškovima izvođenja.

SISTEMSKI IN APLIKATIVNI PRIPOMOČKI
SYSTEMS AND APLICATIONS TOOLS

IZBIRA JEZIKA ZA PODATKOVNO VODENE RAČUNALNIKE

P. KOKOL, M. OJSTERŠEK, V. ŽUMER
TEHNIŠKA FAKULTETA MARIBOR
B. STIGLIC - ISKRA AVTOMATIKA LJUBLJANA

UDK: 681.3.06

POVZETEK - V članku opisujemo lastnosti programskih jezikov, ki so primerni za programiranje podatkovno vodenih računalnikov.

ABSTRACT - Language evaluation for data flow systems. In this paper properties of programming languages suitable for data flow systems programming are given.

1. UVOD

Večina današnjih računalnikov temelji na več kot trideset let stari von Neumannovi organizaciji. Ti računalniki so sekvenčni, opravijo eno operacijo na časovno enoto, imajo en sam procesni element, sekvenčno centralizirano kontrolno enoto, nizek strojni jezik in linearen pomnilnik s fiksno dolžino celic. Da se poveča hitrost takšnim računalnikom, je potrebno povečati hitrost posameznim enotam računalnika. Vendar maksimalna hitrost, omejena s hitrostjo svetlobe, ne bo zadoščala za kompleksne izračune, kakršne lahko pričakujemo v prihodnosti. Zato računalniki prihodnjih generacij temeljijo na arhitekturah, ki dovoljujejo paralelno delovanje in s tem omogočajo, da se mnogo operacij izvrši naenkrat. Od paralelnih arhitektur, ki veliko obetajo, so za nas zanimive podatkovno vodene arhitekture. Delovanje teh je asinhrono, kar pomeni, da vrstni red izvajanja operacij ni določen z nekim centralnim kontrolnim mehanizmom, ampak se operacije izvršijo takrat, ko imajo na voljo vse potrebne vhodne argumente. Podatkovno vodene programe predstavimo kot usmerjene grafe, katerih vozlišča so operacije, povezave pa predstavljajo pretok podatkov, zato jim pravimo grafi pretoka podatkov.

Dobre lastnosti podatkovno vodene arhitekture so:
a) asinhrono proženje,
b) velika možnost paralelnosti,
c) velja pravilo enkratne prireditve,
d) nadzorne ali kontrolne enote niso potrebne.

Slabosti podatkovno vodene arhitekture so:
a) da je potrebno pri vsaki operaciji s polji ali zapisi prenesti celoten podatek, kajti podatkovno vodeni modeli nimajo pomnilnikov, kamor bi te podatke spravljali,
b) da v primeru, ko algoritmi vsebujejo premalo paralelnosti, so te arhitekture manj učinkovite kot kontrolno vodene.

Zaradi naštetih slabosti podatkovno vodeni računalnik ni primeren za splošno namenske računalnike. Njegove dobre lastnosti pa kažejo, da je uporaben za vodenje procesov, robotiko in superračunalnike.

2. PROGRAMSKI JEZIKI

V zadnjih letih so bili razviti mnogi programski jeziki, ki omogočajo hkratno izvajanje programov (Ada, Modula, Concurent Pascal, Edison, razni dialekti Fortrana...). Slabost teh jezikov je, da so jezikovni konstrukti, ki omogočajo hkratno izvajanje programov, le pripomoček programerju, da pokaže prevajalniku, kje so možne paralelnosti. Ti jeziki so nenaravni, kajti prilagojeni so sistemom, na katerih se izvajajo, ne pa načinu, kako človek (programer) razmišlja. Če opazujemo razvoj programske opreme, na eni strani vidimo, da se je pojavila t.i. "softverska kriza", ki bi jo lahko rešili z uvedbo mnogo višjega programskega jezika, ki bi zadovoljeval množico zahtev t.i. "referential transparency" /7/. Na drugi strani, pa razvoj materialne opreme z uvedbo VLSI teži k vse večji paralelnosti in k vse bolj paralelnim arhitekturam. Tudi jeziki, ki bi ustrezali takšnim arhitekturam, bi morali zadovoljevati natanko enake zahteve kot so zahteve "referential transparency".

2.1. Paralelnosti v programskih jezikih

Vidimo, da tako razvoj materialne kot razvoj programske opreme zahteva uvedbo jezikov, ki bi omogočali enostavno (blizu človekovemu načinu razmišljanja) in učinkovito izražavo paralelnosti. Zato si poglejmo, na kakšne načine se paralelnosti pojavljajo v programskih jezikih /14/:
a) eksplicitna paralelnost - pomeni, da je program naravno paralelen (inherentna paralelnost), ali da so za paralelnost dodani posebni jezikovni konstrukti (forall zanke, cobegin itd.),
b) lokalna paralelnost - pomeni, da se paralelnost odkriva v majhnih delih programa (interproceduralno) kot so bloki, zanke itd.,
c) globalna paralelnost - pomeni, da je za odkrivanje paralelnosti potrebna neka globalna intraproceduralna analiza, ki je nepraktična in neučinkovita,
d) neodkriljiva paralelnost - ta nastane zaradi odvisnosti med podatki ali zaradi neučinkovite predstavitve programa. Postopkovni programski jeziki z globalnimi spremenljivkami, stranskimi efekti in nestrukturiranimi jezikovnimi konstrukti (GO TO, aritmetični if stavek) vsebujejo v glavnem neodkriljive paralelnosti.

Paralelnost lahko odkrivamo na treh nivojih:
a) na algoritemskem nivoju (npr. program v zelo visokem programskem jeziku ali posebni paralelni algoritmi) odkrivamo paralelnosti, kadar programski jezik omogoča zadostno abstrakcijo algoritma in vsebuje veliko eksplicitnih paralelnosti,
b) na nivoju izvornega jezika (npr. program v visokem programskem jeziku) odkrivamo paralelnosti s po-

sebnim predprocesorjem. Te paralelnosti zaradi slabe reprezentacije ne moremo odkriti na algoritemskem nivoju,

c) v času izvajanja (npr. program v strojnem kodu) odkrivamo paralelnosti dinamično. To postane nepraktično, kadar se mora paralelno izvesti mnogo instrukcij in torej ni primerno za moderne VLSI arhitekture, ki omogočajo hkratno izvajanje več tisoč instrukcij.

2.2. Lastnosti programskih jezikov

Programske jezike lahko razdelimo, kot so prikazani na sliki 1.

Programi pisani v postopkovnih jezikih imajo naslednje pomembne lastnosti:

a) Uporabljajo model s stanji, kar onemogoča učinkovito paralelno izvajanje in zahteva uporabo arhitekture s stanji.
b) Uporabljajo kompleksne nematematične strukture, s čimer otežkočijo ali celo onemogočijo odkrivanje paralelnosti ter optimizacijo, preslikavo in verifikacijo programov.
c) Uporabljajo globalno okolje, kar povzroča velike odvisnosti med podatki, s čimer omejujejo paralelno izvajanje in dekompozicijo programov.
d) Izvajajo operacije z eno besedo podatka na časovno enoto, tudi če mora biti ista operacija izvedena z mnogo besedami (von Neumannovo ozko grlo).
e) Preslikavajo "pomnilnik" v "pomnilnik", zaradi tega so nefleksibilni in jih je težko sestavljati v obsežnejše programe.
f) Povzročajo veliko semantično vrzel med koncepti visokih programskih jezikov in koncepti obstoječih arhitektur.
g) Omogočajo samo deterministično programiranje, zaradi tega jih je težko uporabljati za reševanje problemov umetne inteligence, ki zahteva heuristično obdelavo podatkov.

V nasprotju s postopkovnimi jeziki pa nepostopkovni jeziki vsebujejo mnogo lastnosti, ki omogočajo izražava paralelnosti, enostavno verifikacijo programov, sestavljanje programov v kompleksnejše programe idr.

Lastnosti posameznih jezikov so podane v tabeli 1, kjer imajo okrajšave naslednji pomen:

S	model s stanji
N	programski jezik vsebuje nestrukturirane konstrukte
G	programski jezik uporablja globalno okolje
P	paralelnost
SE	možnost sestavljanja manjših programov v večje
PR	postopkovni jezik
NPR	nepostopkovni jezik
V	možnost matematične verifikacije programov
R	repetativen - vsebuje konstrukte za ponavljanje (npr. zanke)
ND	omogoča nedeterministično programiranje.

Značilnost definicijskih jezikov je t. i. pravilo enkratne povezave. Zaradi tega pravila so definicijski jeziki zelo primerni za verifikacijo programov in za programiranje podatkovno vodenih računalnikov.

Logični jeziki omogočajo paralelno in logično programiranje. Predvideni so za programiranje računalnikov pete generacije in tudi za podatkovno vodene računalnike.

Objektni jeziki uporabljajo kot osnovo t.i. sporočilo /objekt - model in ne bolj znan operator/ operand - model. V objektnih jezikih definiramo objekte ali razrede objektov (ti imajo skupne lastnosti). Objekti vsebujejo tako deklaracijo (opis) podatkov kot postopke za obdelavo teh podatkov. Če hočemo izvršiti kakšno operacijo, pošljemo določenemu objektu sporočilo, ta izvede ustrezno akcijo ali pošlje sporočilo drugemu objektu.

LISP-ovski in čisti funkcijski jeziki vsebujejo tri vrste struktur: procedure, izraze, spremenljivke in konstante. Glavne značilnosti funkcijskih jezikov so:
- ne vsebujejo ponavljalnih struktur, prireditvenih stavkov in spremenljivk,
- preslikujejo objekte v objekte (teh objektov ne smemo zamenjati z objekti v objektnih jezikih).

Čisti funkcijski jeziki so zelo primerni za programiranje podatkovno vodenih računalnikov kakor tudi za druge arhitekture prihodnosti kot so Magov /8/ računalnik, redukcijski računalnik idr.

Programiranje v redukcijskih jezikih je dosledno funkcijsko in temelji na enostavnih matematičnih konstruktih, ki predstavljajo strukturo binarnih dreves, iz katerih z rekurzivno uporabo sestavljamo kompleksnejše izraze. Redukcijske jezike uporabljamo predvsem za programiranje redukcijskih računalnikov.

Vsi trije tipi jezikov pretoka podatkov (JPP) so si zelo podobni v svojih osnovnih lastnostih. Vsi predstavljajo abstrakcijo grafa pretoka podatkov za neki algoritem. Razlika med grafičnim in visokim JPP je podobna kot razlika med opisom algoritma v psevdokodu in opisom algoritma z grafičnimi pripomočki, kot so npr. diagrami poteka. Kodirni JPP pa je s posebnimi instrukcijami opisan graf pretoka podatkov.

S stališča verifikacije, dokumentacije, preglednosti in obsežnosti (programa, ki obsega samo nekaj vrstic teksta v visokem JPP, bi obsegal več strani in množico povezav v grafičnem JPP) mislimo, da je iz trojice jezikov pretoka podatkov visoki JPP najboljši.

Visoki JPP vsebuje elemente (sintakso) postopkovnih jezikov in semantiko čistih funkcijskih ter definicijskih jezikov. Uporabljamo jih za programiranje podatkovno vodenih računalnikov in za avtomatično programiranje.

3. LASTNOSTI JEZIKA PRIMERNEGA ZA PODATKOVNO VODENI RAČUNALNIK (PVR)

Da bi jezik ustrezal podatkovno vodenim arhitekturam, mora ugoditi zahtevama:

I. Pretok podatkov se mora ugotoviti iz operacij programa.
II. Potek izvajanja mora biti enak pretoku podatkov.

Kadar programski jezik ustreza gornjima zahtevama, mora imeti naslednje lastnosti:

a) lokalnost efektov,
b) ni stranskih efektov,
c) pravilo enkratne prireditve,
d) posebni konstrukti za iteracije,
e) ni nestrukturiranih konstruktov,
f) ni zgodovinsko občutljiv,
g) ni globalnega okolja,
h) ni spremenljivk (imena so le označbe vrednosti)
i) primeren za arhitekture z asinhronim proženjem.

Zgornje lastnosti omogočajo:
- enostavnejšo verifikacijo programov,
- paralelnosti se ugotavljajo interproceduralno (lokalna paralelnost),
- paralelnosti se ugotavljajo na algoritemskem nivoju (program je zelo dobra abstrakcija algoritma),
- jezik je bolj razumljiv uporabnikom in predvsem začetnikom,
- graf pretoka podatkov je naravna preslikava programa,
- ustreza VLSI arhitekturam.

Vrednosti in imena

Jeziki, primerni za programiranje PVR morajo biti vrednostno orientirani, kar pomeni, da v programih uporabljajmo zgolj vrednosti, imena pa vam predstavljajo le označbe teh vrednosti.

Lokalnost efektov

V postopkovnih jezikih je mogoče ista imena za spremenljivke uporabljati v različne namene - v različnih delih programa.

Če ne bi bilo enakosti imen, bi te dele programa lahko izvajali paralelno. Odkrivanje takšnih paralelnosti je mogoče, vendar je enostavneje omejiti delovanje imen le na neko določeno območje programa in kontrolirati vhode in izhode tega območja. Še bolje je, če se izognemo globalnim imenom, proceduram dovolimo le dostop do svojih parametrov in strukturiramo programe v bloke. Lokalnost efektov torej onemogoča odvisnosti podatkov, ki so v različnih blokih.

Stranski efekti

Odsotnost stranskih efektov je nujen pogoj za enakost med pretokom podatkov in potekom izvajanja. Medtem ko lokalnost efektov zahteva le nekatere manjše spremembe v jeziku, odsotnost stranskih efektov zahteva spremembo načina obdelave podatkov. Zaradi tega prepovemo globalna imena in dodamo pravilo, da procedura ne sme spremeniti ničesar - niti svojih vhodnih parametrov. S tem odstranimo enostavne stranske efekte, ostanejo pa stranski efekti, ki nastanejo zaradi obdelave sestavljenih podatkovnih struktur (npr. polja ali zapisi). Pri teh je namreč skoraj nemogoče ugotoviti, kakšne odvisnosti podatkov povzroči sprememba posameznega elementa sestavljene podatkovne strukture. V takšnem primeru je treba vzeti največjo možno odvisnost podatkov, s čimer zelo zmanjšamo možnost paralelnega izvajanja. Temu se izognemo, da obravnavamo sestavljene strukture enako kot skalarne, ter da v procedurah parametre kličemo z vrednostjo in ne z referenco. Posledica tega je, da moramo pri spremembi vrednosti posameznega elementa sestavljene strukture zgraditi celotno novo strukturo.

Pravilo enkratne prireditve

V prejšnjih razdelkih smo se odločili za vrednostno orientiran način programiranja. V tem načinu prireditveni stavek ne pomeni več prireditve vrednosti spremenljivki, ampak zgolj povezavo med imenom na levi in vrednostjo na desni strani prireditvenega stavka. S tem postane program enak sistemu matematičnih enačb, kajti kjerkoli v programu lahko zamenjamo izraz na desni z ustreznim imenom na levi strani, ne da bi s tem kakorkoli spremenili pomen programa. Postavimo še naslednji pravili:
P1: ime se mora povezati prej preden ga uporabimo,
P2: ime lahko povežemo samo enkrat.

S pravili P 1 in P 2 in vrednostno orientiranim načinom programiranja dosežemo, da v programih ni spremenljivk in da je prireditveni stavek kar definicija imena.

Iteracije

Zaradi pravil P 1 in P 2 se pojavijo problemi pri iteracijah, ki uporabljajo prireditvene stavke oblike:

I : = I op J

kjer so:

I ..... iteracijsko ime
J ..... ime
op ..... operator.

Ta stavek je v nasprotju s pravilom P 1, ker zaradi pravila P 2 I uporabljamo prej, preden ga definiramo. Vendar so stavki takšne oblike zaradi iteracij potrebni. Prepoved uporabe nestrukturiranih jezikovnih konstruktov nam omogoča, da z ustrezno prireditvijo while-do zanke omogočimo iteracije, ne da bi kršili ostale lastnosti.

Tako prirejena while-do zanka vsebuje:
(1) deklaracijo in inicializacijo iteracijskih vrednosti,
(2) test, če naj se zanka konča,
(3a) če se zanka konča, definicijo izhodnih vrednosti,
(3b) če se zanka ne konča, izraze, ki generirajo nove vrednosti iteracijskih imen.

Čeprav se iteracijske vrednosti spreminjajo, se to zgodi le v prehodu iz ene iteracije v drugo, tako da pravilo enkratne prireditve še vedno velja, četudi samo v sklopu ene iteracije.

Zgodovinsko občutljivo računanje

Ena pomembnih lastnosti, ki jih mora imeti programski jezik primeren za podatkovno vodene računalnike, je zgodovinska neobčutljivost. Zgodovinska neobčutljivost pomeni, da procedure ne vsebujejo spremenljivk, v katere bi shranile podatke med posameznimi klici procedur. To pomeni, da procedure nimajo zmožnosti "pomnjenja", kar v sklopu z ostalimi lastnostmi predstavlja veliko težavo pri računanju v realnem času.

Prepoved globalnega okolja in nestrukturiranih konstruktov

Uporaba globalnega okolja in nestrukturiranih konstruktov nam oteži ali celo onemogoča odkrivanje paralelnosti in analizo pretoka podatkov.

4. VREDNOTENJE JEZIKOV

V tem razdelku bomo ovrednotili jezike glede na lasnosti, ki jih mora imeti programski jezik za učinkovito programiranje podatkovno vodenega računalnika. Poleg navedenih lastnosti v razdelku 3 smo ocenjevali še dve pomembni splošni lastnosti:
- preglednost programa in
- možnost matematične verifikacije programov.

Pri vrednotenju jezikov upoštevamo naslednje ocene:

+ . jezik ima potrebno lastnost,
Ø jezik lahko ima potrebno lastnost, vendar se to od jezika ne zahteva,
- jezik nima potrebne lastnosti.

Spodnjo mejo primernosti dobimo tako, da seštejemo lastnosti označene s +, zgornjo pa tako, da spodnji meji prištejemo lastnosti označene z Ø.

Tabela 2 vrednoti posamezne jezike z upoštevanjem lastnosti od a do i.

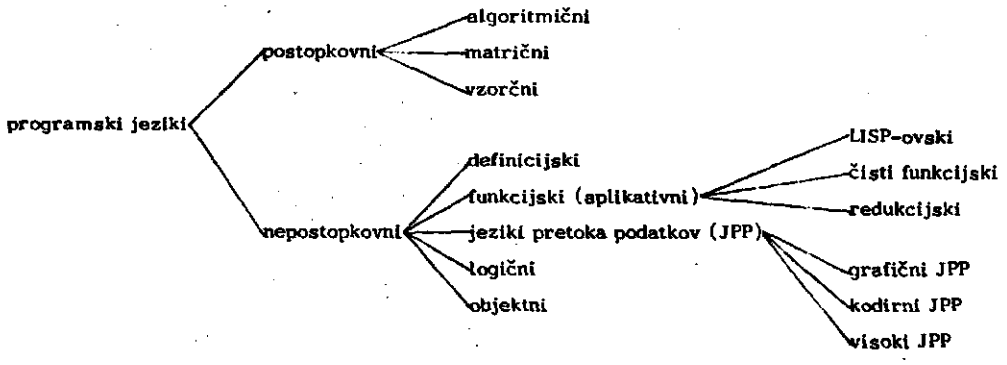
Vrednotenje logičnih in objektnih jezikov ni realno, ker se struktura njihovih predstavnikov (Prolog, Smalltalk) ne ujema popolnoma s shemo našega vrednotenja. Iz tabele vidimo, da čisti funkcijski jeziki, redukcijski jeziki in visoki JPP vsebujejo vse zahtevane lastnosti. Visoki JPP so po sintaksi jezika bližji postopkovnim jezikom kot funkcijski ali redukcijski jeziki in zaradi tega bolj sprejemljivi za uporabnike vajene postopkovnega stila programiranja.

5. ZAKLJUČEK

V članku smo na kratko opisali jezike, ki so primerni za programiranje podatkovno vodenih arhitektur. Programske jezike smo razdelili v več skupin in ocenili njihove lastnosti.

LITERATURA

1. Data - flow computers, IEEE Computer, Vol. 15, No. 2, Februar 1982.
2. Ackerman, W. B. Dennis J. B., VAL-A Value oriented Algorithmic Language, Preliminary Reference Manual, MIT Laboratory for CS, 1979.
3. Japanese Computer Technology Culture, IEEE Computer, Vol. 17, No. 3, Marec 1984.
4. Treleaven, P. C., i.dr., Data - Driven and Demand - - Driven Computer arhitecture, Computing Surveys, Vol. 14, No. 1, Marec 1982.
5. Lerner, J. E., Data Flow arhitecture, IEEE Spectrum, Vol. 21, No. 14,April 1984.
6. Kokol, P., Stiglic, B., Žumer, V., Pretvorba aplikativnega jezika v grafe pretoka podatkov, ETAN 1984, Split.
7. Turner, J. D., Sequin, C., Discusion on Very High Level Languages.
8. Backus, J., Function - level computing, IEEE Spectrum, Vol. 19, No. 8, August 1982.
9. Backus, J., Can Programming Be Liberated from the von Neumman Style?, Communications on the ACM, Vol. 21, No. 8, August 1978.
10. Cox, J. B., Message Object programming, An Evelutionary Change in Programing Technology, IEEE Software, Vol. 1, No. 1, January 1984.
11. O'Keete, R. A., Prolog compared with LISP?, SIGPLAN notices, Vol. 18, No. 5, Maj 1983
12. Fifth Generation Computer Systems, edited by T. Moto - oka, North Holland, Pub. Co., 1982.
13. Berkling, K. J., Reduction Languages for Reduction Machines, Proc. Second, Int. Symp. on Computer Architecture, April 1975.
14. Flynn, M. J., Hennessy, J. L. Parallelism and Representation Problems in Distributed Systems, IEEE Trans. on Computers, Vol. 29, No. 12, December 1980.
15. Hoare, C. A. R., Programing: Soucery or Science, IEEE Software, Vol. 1, No. 2., April 1984.
16. Kokol P., Aplikativni jeziki in njihova interpretacija v realnem času, Magistrska naloga, TF - Maribor.



Slika 1: Delitev programskih jezikov

JEZIK \ LASTNOST	S	N	G	P	SE	PR/NPR	PREDSTAVNIK	V	R	ND
FORTRANski	DA	DA	DA	možna	relativno dobra	PR	FORTRAN, BASIC	slaba	DA	NE
ALGOLski	DA	možno	DA	možna	slaba	PR	ALGOL, pascal	slaba	DA	NE
procesni	DA	možno	DA	možna	slaba	PR	PEARL, LTR	slaba	DA	NE
matrični	DA	DA	DA	NE	DA	PR	APL	slaba	DA	NE
vzorčni	DA	DA	DA	?	?	PR	SNOBOL, RASBOL	slaba	DA	NE
definicijski	možno	NE	možno	inherentna	dobra	NPR	Lucid	zelo dobra	možno	NE
LISP-ovski	DA	NE	DA	"	možna	NPR	LISP	možna	DA	DA
čisti funkcijski	NE	NE	NE	"	zelo dobra	NPR	EP	zelo dobra	NE	DA
redukcijski	možno	NE	NE	"	zelo dobra	NPR	RED	zelo dobra	NE	DA
grafični JPP	NE	NE	NE	"	dobra	NPR	FGL, GPL	NE	DA	NE
kodirni JPP	NE	NE	NE	"	dobra	NPR	/	NE	DA	NE
visoki JPP	NE	NE	NE	"	dobra	NPR	ID, VAL	zelo dobra	DA	NE
logični	DA+	NE	DA	"	možna	NPR	PROLOG, KL-0	možna	NE	DA
objektni	DA	NE	DA	NE	dobra	NPR	SMALLTALK, OBJECTIVE-C	možna	DA	možna

+ na konvencionalnih arhitekturah

Tabela 1: Lastnosti jezikov

* v nekaterih dialektih

jezik \ lastnost	a	b	c	d	e	f	g	h	i	VER.	PREG.	primernost
postopkovni	-	-	-	+	-	-	-	-	-	-	+	2 - 2
definicijski	ø	ø	+	ø	+	ø	ø	+	+	+	ø	5 - 11
čisti funkcijski	+	+	+*	+**	+	+	+	+	+	+	+	11 - 11
LISP-ovski fun.	+	ø	-	+	+	-	ø	-	ø	ø	ø	3 - 5
redukcijski	+	+	+*	+**	+	+	+	+	+	+	+	11-11
grafični JPP	+	+	+	+	+	+	+	+	+	-	ø	9 - 10
visoki JPP	+	+	+	+	+	+	+	+	+	+	+	11 - 11
kodirni JPP	+	+	+	+	+	+	+	+	+	-	-	9 - 9
logični	+	ø	+*	+**	+	ø	ø	ø	+	ø	ø	5 - 11
objektni	+	ø	ø	ø	+	-	ø	ø	-	ø	+	3 - 9

VER. verifikacija
PREG. preglednost

* sploh ni prireditvenih stavkov

** jezik ni repetativen

Tabela 2: Vrednotenje jezikov

FORMATER TEKSTA IZVORNOG PROGRAMA ZA PASCAL

Z.VUKAJLOVIĆ[1], A.ZELE[2], M.OVČINA[3]
1) Elektrotehnički fakultet Sarajevo
2) IRIS - Energoinvest Sarajevo
3) UNIS-INSTITUT Sarajevo

UDK: 681.3.06

KRATAK SADRŽAJ: U radu je opisan formater teksta izvornog programa za programski jezik PASCAL. Izborom odgovarajućih vrijednosti za opcije omogućuje se velika fleksibilnost u obliku uredjenog teksta, koji je uredjen u skladu sa statičkim nivoima ugnježdenja, a prema sintaksnim pravilima jezika.

TEXT FORMATER OF SOURCE PROGRAM FOR PASCAL: In this papaer is described text formater of source program writen in programming language PASCAL. Its rich set of options for formating provides great flexibiliti in formating the text, which is arranged according to static nesting levels and sintactic rules of programming language.

1. UVOD

Kod jezika sa blokovskom strukturom (a i uopšte) vizuelna preglednost programa je dosta bitna ze njegovu čitljivost. S toga, programer u procesu pisanja programa istu povećava na taj način što sintaksne jedinice koje su na višem statičkom nivou ugnježdenja "uvlači" pokazujući na taj način statičku strukturu programa. Medjutim, rijetko se dešava da u programu nema grešaka, što ima za posljedicu da se prvobitna "čitljivost" pogoršava u procesu izmjene programa. Da bi se čitljivost ponovo upostavila, potrebno je tekst izvornog programa urediti u skladu sa blokovskom strukturom jezika, odnosno formatirati ga.

U ovom radu je prikazan jedan takav formater teksta izvornog programa za PASCAL. Nastao je na bazi (dobrih i loših) iskustava u korištenju formatera teksta izvornog programa za OMSI PASCAL [1] , kao i iskustava stečenih u realizaciji formatera za programski jezik EDISON [2]. Karakteristika formatera [1] je da korisnik nema velikih mogućnosti u izboru oblika uredjenog teksta. Iskustva u realizaciji formatera [2] su bila veoma korisna i u datoj realizaciji, ali treba napomenuti, da PASCAL, u odnosu na EDISON, ima sintaksu koja je manje "pogodna" za formatiranje, što je, u krajnjoj liniji, uticalo na nešto drugačiji pristup u projektovanju i realizaciji.

Oblik uredjenog teksta se definiše opcijama, koje su opisane u poglavlju 2 rada. U poglavlju 3 je opisan način realizacije, dok su u dva posljednja navedeni zaključci i literatura.

2. IZBOR OBLIKA UREDJENOG TEKSTA I OSTALE MOGUĆNOSTI FORMATERA

Karakteristika realizovanog formatera teksta izvornog programa je, da programer, izborom odgovarajućih opcija, može uticati na oblik uredjenog teksta. Korisnik opcijama može definisati:
- maksimalnu dužinu linije formatiranog teksta,
- maksimalan broj sintaksnih jedinica u jednoj liniji, posebno u deklarativnom, a posebno u iskaznom dijelu programa,
- način ispisa riječi, rezervisanih i identifikatora,
- broj mjesta za uvlačenje sintaksnih jedinica na višem statičkom nivou ugnježdenja,
- broj mjesta za uvlačenje nastavka sintaksnih jedinica koje nisu mogle stati u jedan red,
- ispis rezervisane riječi END u posebnom redu,
- ispis teksta po stranicama, sa numerisanim linijama, a uz izbor veličine stranice, i
- signalizaciju leksičkih i sintaksnih grešaka, ubacivanjem specifičnog komentara na mjesto greške.

Odredjene vrijednosti za sve opcije su predefinisane, a programer može birati vlastite, definišući ih na sljedeći način:
"/" ime_opcije ":" vrijednost
Ime opcije se zadaje jednim znakom (slovom), a vrijednost nenegativnim cijelim brojem.

Na raspolaganju su opcije L, M, P, D, I, U, N, E i S.

Opcijom L se bira maksimalna dužina linije, koja ne može biti duža od 132 znaka. Predefinisana vrijednost za ovu opciju je 75 (znakova).

Opcijom M se bira način ispisa ključnih (rezervisanih) riječi i identifikatora. Može imati vrijednosti 1,2 i 3 (predefinisano 1), a sa sljedećim značenjem:
- 1: rezervisane riječi se pišu velikim, a identifikatori malim slovima,
- 2: i rezervisane riječi i identifikatori se pišu malim slovima,
- 3: i rezervisane riječi i identifikatori se pišu velikim slovima.

Napomenimo, da sa stanovišta korištenja velikih i malih slova, komentar ostaje neizmijenjen.

Opcijama D i I se definiše maksimalan broj sintaksnih jedinica u jednoj liniji, i to, posebno u deklarativnom a posebno u iskaznom dijelu programa. Predefinisana vrijednost za D je 1, a za I je 2.

Opcijom P se bira mogućnost ispisa uredjenog teksta po stranicama. U tom slučaju, linije izlaznog teksta se numerišu, a vrijednošću opcije se definiše veličina strane. Ova opcija se koristi samo u slučaju kada se uredjeni tekst želi koristiti za dokumentovanje.

Opcijom U se definiše za koliko se mjesta sintaksna jedinica višeg statičkog nivoa ugnježdenja "uvlači" u odnosu na sintaksnu jedinicu (za jedan) nižeg statičkog nivoa ugnježdenja (predefinisana vrijednost 2).

Opcijom N se definiše za koliko je mjesta, u sljedećem redu, uvučen nastavak sintaksne jedinice koja nije mogla stat u jednu liniju (predefinisana vrijednost 8).

Opcijom E se definiše gdje se piše rezervisana riječ END, sa kojom se završava neka sintaksna jedinica (npr. složena rečenica). Nultom vrijednošću opcije se naznačava pisanje END-a u istom redu u kojem je i prethodni dio sintaksne jedinice, a nenultom vrijednošću se zahtijeva pisanje END-a u zasebnom redu.

Opcija S se koristi u cilju signalizacije leksičkih i sintaksnih grešaka. Naime, ako se izabere nenulta vrijednost ove opcije, onda se na mjesto greške ubacuje specijalan karakter, odnosno sekvenca (жжжж).

Opcije se mogu definisati na dva načina:
- u tzv. komandnoj liniji, uz ime izlazne datoteke, i
- u okviru komentara koji započinje sa $, navodjenjem opcija iza znaka $.

Mogućnost promjene vrijednosti opcija, navodjenjem novih vrijednosti u okviru komentara, formateru daje dodatnu fleksibilnost u izboru oblika formatiranog teksta.

Komentar se u opštem slučaju takodje uredjuje, stim da postoji mogućnost kopiranja istog u izvornom obliku, što je naročito važno kod prevodilaca kod kojih se silazak na asemblerski nivo programiranja ostvaruje pisanjem asemblerskog koda u obliku komentara. Kopiranje komentara u izvornom obliku se naznačuje sekvencom $C, na početku istog.

Ako se formatiranje vrši korištenjem nenulte vrijednosti opcije P, na kraju listinga se daje popis korisničkih definisanih procedura i funkcija. Za svaku od njih se daju podaci o:
- broju linije u kojoj je procedura (funkcija) definisana,
- broju linije u kojoj započinje tijelo (iskazni dio),
- tipu (P - procedura, F - funkcija i M - program),
- načinu definisanja tijela (N za normalno definisano tijelo, a F, odnosno E u slučajevima da tijelo procedure nadomješćuje rezervisana riječ FORWARD, odnosno EXTERNAL), i
- imenu procedure.

U stvari, uporedo se daju dvije liste, jedna alfabetski uredjena, a druga prema redoslijedu pojavljivanja. U drugom slučaju, imena procedura (funkcija), definisanih na višem statičkom nivou ugnježdenja, su "uvučena" u odnosu na imena procedura (funkcija) na nižem statičkom nivou ugnježdenja, odražavajući na taj način statičku strukturu programa.

3. REALIZACIJA FORMATERA

Formater je realizovan u 4 faze:
- faza prilagodjenja ulaznom tekstu programa,
- faza leksičke analize,
- faza sintaksne analize, i
- faza prilagodjenja željenom obliku izlaza.

Faza prilagodjenja tekstu ulazne datoteke je, u odnosu na realizaciju prikazanu u [2] , pojednostavljena i svodi se na čitanje jednog po jednog znaka ulaznog teksta, uz obradu separatora kraja linije, pri čemu se mora sačuvati samo posljednji simbol jezika (u obliku niza koji se sastoji od jednog ili više znakova, maksimalne dužine 100).

Zadatak leksičke analize je da prepozna odredjene bazne jedinice izvornog jezika, tzv. simbole ili leksičke jedinice jezika, a koje se koriste kao "ulaz" u sintaksnu analizu. Kao simboli jezika se mogu pojaviti: identifikatori, literali (različite konstante) i terminalni simboli (rezervisane riječi, operatori i drugi separatori). Pored toga što treba da prepozna "klasu" simbola, leksički analizator ili tzv. skaner treba da dâ i druge informacije o prepoznatom simbolu. Npr. ako je skaner prepoznao identifikator, treba da dâ informaciju o kojem je identifikatoru riječ, za terminalne simbole koji je to terminalni simbol itd. U procesu leksične analize se vrši i obrada komentara, u okviru kojih se mogu pojaviti definicije novih vrijednosti opcija.

Zadatak sintaksne analize je da prepozna odredjene sintaksne konstrukcije, tzv. sintaksne jedinice. Sintaksne jedinice se formiraju prema odredjenim pravilima, definisanim sintaksom jezika. Sintaksna analiza se vrši sintaksnim analizatorom (gramatički analizator). Ulaz u sintaksni analizator je niz simbola prepoznat u leksičkoj analizi, a izlaz je sintaksno ispravan (korektan niz simbola). U realizaciji je korištena analiza odozgo prema dole, i to metoda rekurzivnog upuštanja.

U sintaksnoj analizi posebna pažnja posvećena je aspektu oporavka od grešaka. On obezbijedjuje sinhronizaciju izmedju procesa analize i skaniranja simbola. Sinhronizacija se vrši: na početku sintaksne jedinice, odnosno procedure kojom se data sintaksna jedinica analizira (prepoznavanjem "startnih" simbola, sa kojim data sintaksna jedinica može započeti), na izlasku iz procedure (prepoznavanjem regularnih "sljedbenika", definisanih odgovarajućim konstantnim parametrom procedure), a i u samoj proceduri (prepoznavanjem karakterističnih simbola jezika).

Faza prilagodjenja željenom obliku izlaza, realizovana je na način sličan realizaciji u [2]. Odredjene razlike (zbog "nepogodnosti" sintakse PASCAL-a) se pojavljuju kod izbora početnih pozicija pojedinih ugnježdenih sintaksnih jedinica. Uglavnom, u istoj liniji se ne mogu nalaziti sintaksne jedinice različitih statičkih nivoa ugnježdenja, takvih da je posljednja od njih nižeg nivoa ugnježdenja, da nije drugi dio složene sintaksne jedinice. Npr. na izlazu se ne može pojaviti:

```
BEGIN I:=1; J:=2;
K:=3 END; I:=I+1;
"Pravilni" oblik iskaza je:
BEGIN I:=1; J:=2;
  K:=3 END;
I:=I+1;
```

4. ZAKLJUČAK

Formater izvornog programa programski jezik PASCAL je realizovan na računaru PDP-11/23, pod operativnim sistemom RSX 11/M u Laboratoriji za elektroniku i računarsku tehniku, a u okviru projekta DIPSY (kojeg finansira SIZ Nauke BiH). Napisan je u programskom jeziku visokog nivoa (PASCAL), pa je stoga veoma prenosiv.

Osnovne karakteristike realizovanog formatera se ogledaju kroz njegove znatne mogućnosti u izboru oblika izlaznog teksta, i njegovoj fleksibilnosti (mogućnosti mjenjanja oblika uredjenog teksta u samom procesu formatiranja). Pored toga, formater ima veoma dobar mehanizam oporavka od grešaka (osobito u odnosu na [1]). Za razliku od formatera [1] , dati ne formira tabelu kros-referenci procedura (sa stanovišta definisanja i korištenja), što se ne mora tretirati kao njego nedostatak (u pitanju je formater teksta, a ne analizator kros-referenci). Ono što ublažava ovaj nedostatak je činjenica da se, kod korištenja opcije P) daju liste, jedna alfabetski uredjena, a druga prema redoslijedu pojavljivanja definisanih procedura i funkcija.

Iskustva stečena u realizaciji datog formatera će, najvjerovatnije, biti veoma korisna kod projektovanja i realizacije sintaksno orijentisanog editora za programski jezik PASCAL.

5. LITERATURA

1. FMT - Formater teksta izvornog programa za OMSI Pascal
2. M.OVČINA: Formater teksta izvornog programa za EDISON, diplomski rad, Elektrotehnički fakultet, Sarajevo, 1985.
3. A.ZELE: Formater teksta izvornog programa za PASCAL, diplomski rad, Elektrotehnički fakultet, Sarajevo, 1985.
4. K.JENSEN, N.WIRTH: PASCAL User Manual and Report, Springer Verlag, 1975.

OPERACIJSKI SISTEM CDOS

Golja Tatjana, dipl.ing. ISKRA DELTA
Kejžar Bogdan, mag. ISKRA DELTA
Dolenc Janez, dipl.ing. ISKRA DELTA

UDK: 681.3.06

V članku je podan pregled operacijskega sistema CDOS. To je sistem z izvajanjem v realnem času, več poslov hkrati in je eno ali večuporabniški mikroračunalniški sistem. Je neodvisen od CPU in periferije. Omogoča programiranje, ki je postalo popularno zaradi sistemov tipa UNIX. Zagotavlja IBM PC DOS 2.1 aplikacijsko okolje.

The following article summarizes CDOS features. CDOS is a real-time multitasking operating system, designed for single-and multiuser microcomputer systems. It is CPU and peripheral independent. Its programming interface allows programming that has become popular through UNIX type systems with standard I/O and pipes. Provides IBM PC DOS 2.1 application enviroments.

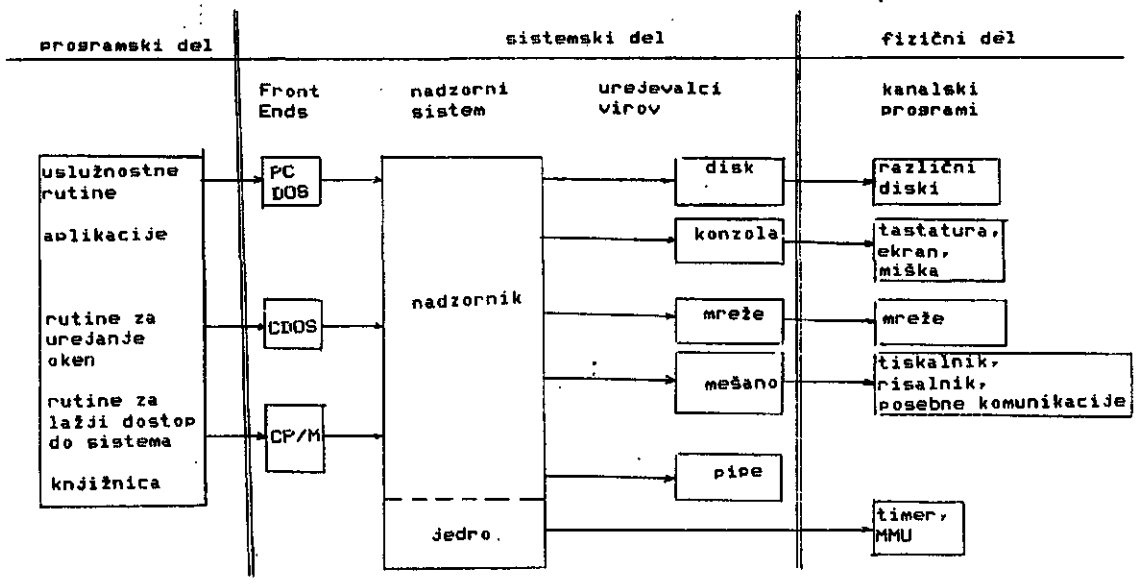
1. Uvod

V članku je kratek pregled operacijskega sistema Concurrent DOS firme Digital Research. Sistem se lahko izvaja na 16 ali 32-bitnih mikroprocesorjih. Za svoje delovanje potrebuje vsaj 512 Kb RAM. Nudi možnost uporabe 2Gb velikega diska.
Pisan je tako, da je neodvisen od mikroprocesorja in periferije. Sistem je večuporabniški ali enouporabniški večprocesni in v realnem času (multi-user ali single-user, multitasking, real-time).

Ob načrtovanju sistema so izbrali kriterije, ki naj jih sistem izpolnjuje:

- funkcionalna kompatibilnost z IBM PC DOS 2.0,
- kompatibilnost diskov z IBM PC DOS 2.0,
- funkcionalna kompatibilnost s CP/M-86 3.1,
- konverzijski programi za CP/M-86 3.1,
- možnost priključitve standardnih terminalov,
- možnost dinamičnega nalaganja kanalskih programov,
- orodja za delo z okni (windows),
- uporaba grafike v oknih,
- izvajanje v realnem času (real-time),
- asinhroni dogodki,
- neodvisnost od centralnih procesnih enot (CPU),
- zaščiten pomnilnik,
- več-uporabniški ali eno- uporabniški sistem,
- vmesnik za delo z računalniškimi mrežami (dodatek),
- vmesnik za grafične rutine GSX (Graphics Extension),
- možnost deljenega dostopa do sistema datotek,
- bolj pomembne so performanse sistema kot velikost,
- absolutni minimum notranjega pomnilnika je 128 Kb,
- minimum notranjega pomnilnika za normalno delovanje je 512 Kb,
- modularnost za delo krmilnih aplikacij v realnem času,
- orodja za pisanje aplikacij, da so neodvisne od strojne opreme (terminal, disk, CPU),
- orodja, namenjena vključevanju interaktivnih naprav (terminali, grafične naprave, tastature),
- programsko orodje naj bo tako kot na operacijskem sistemu UNIX (standardni vhod, izhod, pipe),
- obstoječi programi naj izkoriščajo nove možnosti v čim večji meri,
- omogoča normalno delovanje v primeru napak na vhodno/izhodnih napravah (full error recovery),
- nudi naj orodja za pisanje aplikacij, ki uporabljajo več procesov (multi-taskins) in več oken (multi-windowins),
- omogoča naj uporabo 16-bitnih tujih naborov znakov (tudi KANJI),
- omogoča naj seneracijo sistema brez neželjenih značilnosti,
- sistem naj bo zanesljiv,
- sistem naj bo hiter.

2. Organizacija sistema



slika 1: struktura operacijskega sistema

Kot je vidno iz slike 1, je Concurrent DOS zgrajen iz treh ločenih delov:

- programski, ki vsebuje uslužnostne pomožne rutine, rutine za lažji dostop do sistema, aplikacije, rutine za urejanje oken (window management) in knjižnico rutin,ki uporabljajo sistemske rutine;
- sistemski del, ki vsebuje vse rutine operacijskega sistema, neodvisne od strojne opreme in uporabniških vmesnih rutin. Je specifičen za posamezne centralne procesne enote (CPU). Vsebuje rutine, ki omogočajo izvajanje programov z operacijskih sistemov CP/M ali PC-DOS (Front-Ends), rutine nadzornega dela sistema (Supervisor) in rutine, ki pripravijo vse za delo s periferijo (Resource Managers);
- fizični del pa predstavlja vse rutine, ki so odvisne od strojne opreme. To so kanalski programi (device drivers).

Vsi programi, ki jih sistem naloži z diska, tečejo neodvisno drug od drugega. Vsak ima svoj del uporabniškega pomnilnika.

Procesor Front End omogoča izvajanje programov, pisanih za IBM PC DOS 1.1 ali 2.n, sistem, za CP/M-86 ali CP/M-68K operacijski sistem. Front Ends procesorji uporabljajo nadzorni modul. Ta kliče del operacijskega sistema, ki dela s periferijo (Resource Managers). Nadzorni modul določi prave rutine za delo s periferijo. Jedro sistema ureja procese, dela s pomnilnikom, olajša komunikacije in delo z mrežami. Kontrolira posebne rutine pri segmentaciji pomnilnika. Osnovna naloga jedra je dodeljevanje virov procesom glede na njihovo prioriteto. Če ima več procesov enako prioriteto, se izvajajo zaporedno tako, da se vsak izvaja določen čas (način round-robin).

2.1. Urejevalci virov (Resource managers)

Rutine za logično delo z diskom, konzolo, mrežo, pipami, tiskalnikom, risalnikom so zbrane v urejevalcu virov (Resource Manager). Ta predstavlja vmesnik med nadzornim modulom (Supervisor) in kanalskimi programi (Device Drivers) za vsako napravo.
Urejuje sistem datotek na disku, ki je lahko kar PC/DOS 2.n hierarhični sistem datotek (Disk Resource Manager).

Urejuje fizične konzole, kar vključuje ekran, tastaturo in miško (Console Resource Manager). Vsaka t.i. konzola ima lahko eno ali več virtualnih konzol. Te so abstraktne naprave, ki se obnašajo tako kot fizične naprave. Aplikacije kreirajo virtualne konzole s klicem nadzornega modula, ki dela z vsemi konzolami kot z datotekami, le da virtualno konzolno datotek uporablja le za kontrolo konzole ne pa za izvajanje vhodno/izhodnih rutin.

Proces, ki kreira virtualno konzolo, ima edini tudi kontrolo nad njo (npr. spreminja velikost oken, vsebino oken).

Kot opcija so rutine, ki urejujejo mreže (Network Resource Manager).

Urejuje medprocesne komunikacije in sinhronizacije (Pipe Resource Management), kar poteka s pomočjo datotek v pomnilniku. Uporabljene so za prenašanje sporočil enega procesa v drugega ali za sinhronizacijo aktivnosti.

Urejuje delovanje risalnikov, tiskalnikov, portov in komunikacijskih naprav (Miscellaneus Resource Manager). Večinoma predaja klic procesa pravemu kanalskemu programu.

2.2 Kanalski programi

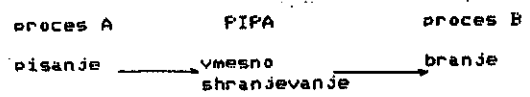
Kanalski programi so vmesnik med sistemov in fizičnimi napravami.
Kanalski program kontrolira več naprav istega tipa (npr.: enega ali več diskov).

Concurrent DOS omogoča menjanje strojne opreme z dinamičnim vključevanjem kanalskih programov med delovanjem sistema. Kanalski programi so lahko stalno v sistemu ali pa so izbrani in naloženi ob nalasanju sistema.

3. Delovanje sistema

Sistem obravnava naprave s pomočjo sistema datotek (file system).
Vsaka datoteka, s katero sistem dela, dobi številko (file number). Dostop do datoteke (branje ali pisanje) se izvaja preko številke. Tak način omogoča kontrolne bloke, neodvisne od tipa datotek in branje, pisanje, neodvisno od naprave. Nadzorni modul tako na enostaven način ugotovi, kateri interni modul je zahtevan.
Sistem ima globalno tabelo odprtih datotek. Standardne številke datotek so:
0 - standardna vhodna datoteka ("stdin"),
1 - standardna izhodna datoteka ("stdout"),
2 - datoteka napak ("stderr"),
3 - ukazna datoteka, od koder je bil program naložen.
Te datoteke so odprte, še preden se izvede prva instrukcija programa.
Dostop do datotek je sekvenčen ali poljuben.
Datoteke so zaščitene z uporabnikovo in skupinsko identifikacijo. Ko se uporabnik priključi na sistem, sistem zahteva geslo (password).

Možna je tudi zaščita celotnega diska.



slika 2: uporaba pip

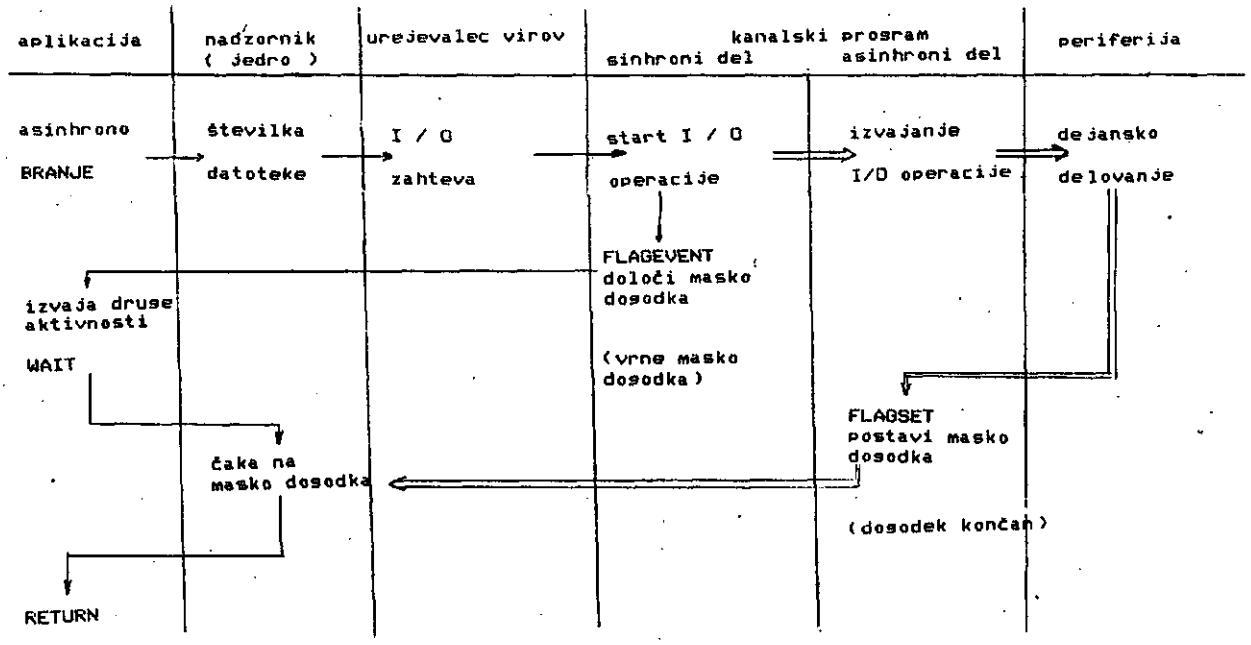
Procesi lahko uporabljajo pipe. Proces A piše v vmesni pomnilnik. Če je podatkov preveč, proces A čaka, da proces B prebere podatke in mu tako dovoli nadaljevanje pisanja. Tudi pri branju sta procesa sinhronizirana.

Slika 3 ilustrira asinhrono branje. Ko aplikacija odpre datoteko, navede njeno ime, v katerem je določena tudi naprava, kjer je datoteka (device:pathname). Sistem vrne karakteristično številko datoteke, s pomočjo katere bo aplikacija komunicirala z nadzornim modulom. Ta izbere ustrezen urejevalec virov (Resource Manager), ki s pomočjo številke datoteke izbere pravo enoto.

Kanalski programi vsebujejo del kode, ki skrbi za sinhrono delovanje in del, ki skrbi za asinhrono delovanje. Sinhroni del je odgovoren za startanje I/O dogodka in za vrnitev v urejevalec virov (Resource Manager), ki je klical kanalski program. Asinhron del izvede I/O operacijo in obvesti sistem, ko konča.

Sistem ureja asinhron vhodno/izhodne operacije s pomočjo zastavic. Prekinitvene rutine (Interrupt Service Routines) kličejo asinhrone servisne rutine, ki uporabljajo sistem zastavic.

Možno je tudi delo s periferijo brez prekinitev (polling).



slika 3: doseg aplikacije do periferije

4. Zaključek

Concurrent DOS je operacijski sistem, ki upošteva in se zgleduje po znanih operacijskih sistemih od primitivnega CP/M do popularnega UNIX-a. Je popolnoma kompatibilen z IBM PC DOS 2.n operacijskim sistemom.

Concurrent DOS je operacijski sistem za delo v realnem času, hkrati se lahko izvaja več procesov, podpira eno ali več-uporabniške računalniške sisteme.

5. Literatura

Digital Research, Inc., Concurrent 4.0 Operating System:
- External Specification, 7.12.1984,
- System Guide, 4.2.1985,
- Programmer's Guide, 25.1.1985,
- User's Guide, 28.1.1985;

REQUIREMENTS ELICITATION BY RAPID PROTOTYPING

Roland T. MITTERMEIR

Institut für Informatik
Universität für Bildungswissenschaften Klagenfurt
A-9020 KLAGENFURT AUSTRIA

UDK: 681.3.01

ABSTRACT

The paper discusses the role of prototyping for developing software. It emphasizes the role of prototypes for elicitation and clarification of requirements. Various aims for "prototyping" will be identified. On this basis, requirements for a requirements engineering environment to support rapid prototyping will be established. The paper concludes with a discussion about alternatives of prototyping environments for evolvable prototyping.

PROTOTYPING / DOES THE ANALOGY HOLD?

Prototyping is a word which comes from classical engineering disciplines. There, a "Prototype" is defined to be "the first full-scale model of a new type or design of furniture, machinery, or vehicle", or, in a more general sense, "an individual that exhibits the essential features of a later individual or species - precursor; an individual, quality, or complex that exemplifies or serves as standard of the essential features of a group or type - exemplar; an original on which a thing is modeled - pattern" (definitions according to Websters Third New International Dictionary, 1976).

In classical engineering disciplines the justification for prototyping can be found in the nature of industrial mass-production processes. Before a new product is placed on the assembly line, one better tests it as carefully as possible. Thus the high setup costs for mass production are incured only for a version of the product which will most likely succeed on the market.

In engineering disciplines, such as civil engineering or architecture, where products are not destined for mass-production, prototyping is not common. These engineers call their small-scale versions of the final product "models". But even in mechanical engineering, the emphasis on prototyping is declining. Better insight into the subject area and the availability of modeling software are the main reasons why mechanical engineers do no longer classify a design group which stresses the development of a high number of prototypes as "effective". They reserve this quality attribute rather for those groups which have a prototype developed only after extensive computer simulations (models) demonstrated the technical feasibility of a new design idea [13].

In the light of this evidence, one might ask, whether the term "prototype" is not a misnomer in the context of software development. Shouldn't software rather be produced according to a well disciplined, phase structured development cycle? If so, wouldn't it be appropriate to demand a properly done, formal requirements analysis culminating in a clear specification, followed by a design phase based upon this specification? The validations and feasibility checks on this design could be accepted as a substitute for the computer-simulations mechanical engineers perform before hand-crafting a prototype. Wouldn't a software-prototype be superflous, because there are no high setup costs in software mass-production?

After all, what signifies the attribute "rapid" in conjunction with prototyping - as e.g. suggested by the name of a workshop of ACM-SIGSOFT [23] and as part of the title of various publications? In classical engineering disciplines, the prototype being the very first item of a new product is usually made by a pre-industrial production process; it is actually hand-crafted. Compared to the industrial mass-production process on the conveyor belt this is very slow.

In spite of the differnces in the production process for software and for tangible products, the answer to the above questions is: There is a place for prototyping in software development, and, if prototyping is done, the protoype is better developed rapidly!

The need for prototyping can best be seen when considering programs as components of the system which constitutes their operating environment. To aid this perspective, Lehmann [11] classified programs into three categories, S-type for specifyable, E-type for embedded and P-type as the intermediate class.

For the category of S-type programs, a clean mathematical specification might suffice to define the behaviour of the system to be built. Therefore, one might assume that prototypes would be superflous for them. However, Swartout and Balzer [24] gave counter examples which showed a definite intertwinning of specification and implementation even for relatively simple programs.

For the other categories of Lehmans classification, P-type and E-type programs, the difference in perception of requirements before and after introduction of a new system clearly prohibits to base system development solely on paper and pencil requirements analysis. For such systems, requirements analysis will not only be concerned with software aspects. It will be rather "systems requirements analysis". However, at the early stages of a project, the boundaries between hardware, software and organizational aspects are not yet sufficiently evident. Therefore, requirements analysis has to deal with the complete needs structure and must not recourse to specifyable requirements only.

PURPOSE OF PROTOTYPING

Though by now, a substantial fraction of the software engineering community does agree that the classical development life-cycle with its clear-cut sequence of analysis-design-implementation-(test) is not an answer to all open questions, and in spite of the fact that a majority of this fraction might see prototyping as a possible route for escape, there is little agreement about what a prototype really is.

Some authors try to find a definition by focussing on whether a system is built to be thrown away (like the prototypes of our colleagues in mechanical engineering) or whether its mayor parts are kept but gradually changed until the developer or customer is sufficiently content with the result to call it no longer "prototype" but "product". ([20] gives a detailed account on this divergent oppinions.) I do not know, whether Leonardo da Vinci called a painted canvas after it has been completed for the first time a prototype and called it a painting only after he had repainted parts of the background and added a few strokes of light color to the face of the main subject in a portrait, but I doubt that this distinction matters too much for a clear terminology. It is rather the difference in intent - and hence the difference in investment, involvement and method of development - whether something should be called a prototype or a failed first version of the final product.

Prototyping does have its own methodology. This methodology can be compared to that for conducting a scientific experiment. The first step in a well designed scientific experiment is to clarify its objectives, i.e. which currently open questions should be answered after the experiment has been completed. Likewise, with prototyping (as with modeling) the first step to be taken is to clarify the purpose of the prototype. Should it serve the developer to reduce the technical risk, - (e.g. to answer questions which remain open after queuing network modeling for performance analysis or after analytical determination of certain reliability attributes) or should it rather serve to give the user an idea about what systems analysts talk in the process they call "requirements analysis" and "user participation?" These are two completely different questions, demanding a very different approach to prototyping. We conjecture that these differences completely overshadow the differences between evolvable and throw-away prototypes.

Since this paper deals with new ways to approach requirements analysis, it focusses on prototypes for aiding the interaction between users and analysts. The aim of such prototypes has been termed as "reduction of the application risk" (ACM-SIGSOFT Workshop on Rapid Prototyping). Prototypes for reducing the "technological risk" can rather be seen as *instruments to support the design process.*

TERMINOLOGY

Before addressing the issue of prototyping environments for reducing the application risk, we should take stock of the available weaponry for requirements elicitation. This is necessary to avoid terminological fuzziness, but also for methodological concerns. Even a rapid prototype of a complete system will be costly to produce and to study. Therefore, it has to be justified by its expected results.

One of the first questions to be asked is whether a mock-up model of the user interface would suffice. Quite often, the man-machine interface is what concerns users most. Therefore, this aspect of a system should be studied most carefully. Giving the eventual users a chance to feel home with the "surface" of a new system will most likely help them to overcome acceptance problems. However, to study different ways of user-system interaction, one does not need to prototype the complete system. A mock-up user interface, driven by some simulating device will suffice.

If requirements regarding the extent of the system are unclear, incremental development will be the methodological answer. One firstly builds the system with reduced functionality (perhaps in a rapid-prototyping stile of development). Later one adds disputed features as prototypical appendices to the truncated system about which different representatives of the customer organization could not agree whether they would be worth their prize. After the users have played with them, these features can be either brought up to product-quality standards or discarded at little expense.

But what, if the situation to be prototyped is too comprehensive to be captured by a prototypical implementation? Assume a project introducing new technology into an area which is not completely under the control of the corporation planning to make this step into the future. Introduction of telephony or of electronic banking might serve as historical examples of such large steps where complex societal interactions must be anticipated to assess the merit of moving ahead. Eventuelly, a site for prototypical installations should be choosen before such drastic changes are fully implemented. But prior to conducting a "field experiment" with a prototype, the innovator might want to conduct a "laboratory experiment" in the seclusion of his own organization. Such a "laboratory experiment" will lack sufficient reality to call its carrier a "prototype". The term "scenario", introduced by [6] and related to prototyping in [8] and [15] might serve much better to describe such situations.

The reflections conducted above suggest the following definitions for the arsenal of requirements analysis tools:

INTERVIEW: A question - answering process to elicit requirements. It can be conducted in group sessions as well as in individual communication. Various techniques exist to effectively conduct such requirements analysis interviews and to capture their results in a formal or semi-formal notation.

DOCUMENT ANALYSIS or OBSERVATION: Instead of a human partner, the analyst collects information from written sources (secondary data) of the organization or by direct observation. The results of this process should be verified by interviews or by some other requirements elicitation technique and documented in the notation used for the main requirements documents.

SCENARIOS: These are projections of events or situations a user would experience when operating the new system or acting in the changed environment [8]. Scenarios are different from prototypes in so far, as the later are "full-scale operating models" or "first originals" and, therefore, part of the future reality, whereas scenarios are a mechanism to create an illusion about the future. A broad range of alternatives exists to create such illusions. They range from verbal descriptions over movies to plays [9] depending on the stage of a project and the aim of the scenario.

The purpose of scenarios to acquaint clients (or experimental subjects) with alternative future worlds suggests that scenarios should always be presented in plurality [15]. Giving people a choice will change them from passive, head nodding spectators to active participants of a requirements eliciting task force.

Further, it should be noted that the applicability of scenarios is not restricted to requirements analysis. They might be used as profitably for objectives analysis. During the objectives analysis phase of a project one does not yet search for problems and their solutions (the "WHAT" of analysis as opposed to the "HOW" of design); the question is rather "WHY" certain aims and objectives are worthwile to be pursued in the light of scarce resouces and a constantly changing environment [17].

EXECUTABLE REQUIREMENTS MODELING: This term and the term "operational requirements modeling" has been coined by Zave to describe a system which models the interaction of components of real time systems [26]. In Zaves proposal, exchange functions are used to study the dynamics of embedded systems. By displaying the dynamic nature of a system, executable models will show users the consequences of some needs statements they expressed. In this capacity, executable models can be seen as a particular kind of tools to show scenarios. However, due to their strict nature, the primary application of executable requirements models will be to validate the consistency and completeness of timing requirements for software components of embedded systems or to study the effect of design decisions related to process communication. Therefore, executable requirements models serve rather to reduce the technological risk than the application risk. Hence, I will not discuss this approach here any further.

SIMULATION MODELS are another kind of "executable models", since they also serve to study the dynamic behaviour of systems. However, to discuss them in conjunction with prototypes seems to blur the notion of "prototype".

MOCK-UP USER INTERFACES: They present the first terminological problem, because they are real in the sense that users could actually work with them - though in a restricted way - and because they might run already on the hardware eventually used for the interface, possibly even in conjunction with some interface software eventually used. However, mock-up interfaces are only a window to software which is technically not representative for the system to be built. A simulator or an interpreter of exchange functions might serve to fake the functionality of the system which should actually carry out the required tasks.

From the point of view that the real user interface might evolve from such a mock-up interface, the mock-up version might qualify as a kind of prototype. However, not as a prototype for the system to be built but only as a prototype of the interface.

PARTIAL PROTOTYPE: As mentioned above, the user interface is only one aspect of discussion during requirements analysis. For other questions, one might as well construct a system with reduced functionality or a system which lacks some other important features of the final product.

For a clear terminology such a truncated system should not be unconditionally called a prototype. Depending on the aims when building it, one might either call it an incomplete version of the system in the process of incremental development or a prototype of the features under investigation. In the later respect, it will be a mock-up model of the features under investigation - a complete analog to the mock-up user interface.

INCREMENTAL DEVELOPMENT: This is clearly distinct from prototyping in the sense that each increment delivered is ment to constitute a part of the final product. Therefore, putting aside those parts of the code which serve "scaffolding"-purposes, everything delivered should remain unchanged. Thus, while there is a notion of evolvement of the complete product, this notion does not hold for the individual increments. If "evolutionary selection" decides against an increment, this increment should be considered a failure.

Obviously, incremental development has significant impacts on the process of requirements analysis. In addition to the overall analysis that served as the basis for the initial design of the complete system - which formed the basis for the partitioning and scheduling decisions related to the increments - a sequence of complete analysis-design-implementation mini-cycles can be conducted for each increment. This has the advantage that users can phrase the requirements for later increments already based on the experience they gained in operating the incomplete system. Therefore, the decisions regarding partitioning and scheduling must not be based exclusively on the grounds of system design. Considerations from requirements analysis methodology should as well be considered when these decisions are taken.

SYSTEM PROTOTYPE: Having terms for all those partial or non-real and non-evolvable versions or models of a system, we can use the word prototype in the context of software engineering in the same restricted sense as classical engineers do. They would for instance call a wooden car built for tests in the wind tunel a model and call only the first product of a kind which later on should be replicated on the assembly line a prototype. However, the purpose of building the prototype is not to sell it but to test it. Since testing serves to uncover faults, there will be no warranty on a prototype. (In classical engineering disciplines, users will hardly ever see a prototype. What they get confronted with are products of the alpha-series which come with an excellent service instead of a warranty.)

Taking this analogy, one can define a prototype of a software (sub-) system as:

> "an operating version of that (sub-) system that has all necessary characteristics of the final product. It differs from the final product only in so far as both, developer and user, understand that there is no commitment on the part of the developer with regard to qualitative aspects of this (sub-) system prototype."

This definition lacks a few attributes which are very much under debate. These are: "first", "rapid", "throw-away" and "evolvable". I will address them in the next paragraphs.

If "first" would be part of the definition, what about the modifications of this "first", which will logically become the "second", "third", ... and so on. Further, "first" would require a definition of what is ment by "new". I would rather focus on intent than on version number. If a system is released without commitment but with the explicit aim of studying its behaviour or receptance under ordinary operating conditions, it should be called a prototype, irrespective of whether it is all new or just a new version of a system that is already operating but which has been subjected to substantial modifications.

As mentioned above, "rapid" is not a characteristic of the production process of the tangible prototypes built in classical engineering disciplines. However, due to the nature of the software production process, the fact that there is no commitment regarding those qualitative features which are not to be tested, makes the production process a rapid one, even without any special prototyping aids. (C.f. the remarks in [1] regarding the productivity figures for non-delivered lines of code.) It will be more rapid though, if a "prototyping language" is used or if a special prototyping environment can be utilized.

"Throw-away" or "evolvability" is secondary to the purposes of the prototype. It might be instrumental to the experimental design related to prototyping though. The current discussion on software reusability and adaptability as stimulated e.g. by the STARS program of the U.S. Dept. of Defence [5] clearly shows that the term "throw away" is illdefined for software as long as it is not qualified. (E.g. what is thrown away when rewriting an algorithm expressed in Pascal, in C, or when porting a program from one operating environment to another?) One might argue a prototypical system written in LISP has to be thrown away if the specification demands the actual product to be written in COBOL (Would the LISP-system then satisfy the above definition or should it be called a model?) or, on the other extreme, one might argue that the pattern of interaction as expressed in a scenario is kept and incorporated into the final system.

In this paper, the notion of evolvability will be reserved for those prototypes, where modifications on the code level allow to tune the system by local improvements in such a way that the design of the prototype is fully represented in the final product and major portions of code remain the same.

REQUIREMENTS FOR PROTOTYPING ENVIRONMENTS

Classical requirements analysis is an interviewing task, supported by paper and pencile. Therefore, the main emphasis on a requirements engineering environment in the classical sense will be on storing and cross-referencing stated requirements as well as on clerical and managerial aspects. Aspects to be supported are for instance keeping track of the relationship between sources of requirements information and the requirements expressed, keeping track of changing requirements within the process of requirements elicitation and keeping track of the progress of the requirements analysis phase.

Since the documentation of requirements is only abstract (in prose, graphical, or formal notation; but not in terms of tangible entities of the world of the future application or of the future system), verification of a specification is one of the main problems in requirements analysis. A requirements engineering environment might support this verification process for instance by providing a simulator for exchange functions, or by a connection to some formal completeness- or consistency checker.

In a requirements engineering environment based on prototyping, the role of verification will not be as prominent, because the prototype, even if consisting of intangible software, will constitute a tangible entity of the users environment - it is not a document but a reality. Therefore, the product of a requirements interview should be mainly scrutinized for its validity.

Exploring the validity of requirements by examining a prototype for its suitability is methodologically comparable to testing code for its correctness. Therefore, some properties needed with prototyping environments are comparable to the managerial properties demanded from test environments.

Key aspects of a prototyping environment which have an analog in testing environments are features

- to store information about the test design,
  - which questions should be addressed in the prototyping experiment,
  - who are the testers;
- for version control,
  - what versions of a prototype have been built/studied already,
  - how do these versions differ;
- to document the results of the experiments,
  - which questions have been addressed already with a version, which ones are still open,
  - support of "regression testing", i.e. which experiments must be replicated with a new version of a prototype;
- for operations support,
  - help in providing test data.

In spite of the relatedness between testing and prototyping, some differences in the task require differences in the support environment. One of these differences is, that pinning down the testers is less important in testing than in prototyping. An "industrial" testing process should be largely independent from the person conducting the test-runs; not so a prototyping experiment.

By the nature of the approach, a methodology based on prototyping will follow a participative approach to requirements engineering. Because in any sizeable project, various persons will be involved in this participative endeavour, a major fraction of requirements analysis costs will not be spent in the (rapid) production of a prototype or a new version thereof, but in playing around with it, i.e. testing it.

To keep these testing costs as low as possible, the experimental design should clearly state, which experiments (test cases) are to be conducted with which experimental subject (future user). It might be though, that this assignment of experiments to subjects has to be updated in the course of the prototyping experiment.

Another aspect of conducting a prototyping experiment as efficiently as possible is careful choice of test data. The environment which provides operational support should be instrumental in this respect. However, this feature will also be slightly different from the related feature of a testing environment. For testing code, test data can be determined from the structural properties of the code (c.f. all sorts of "white box testing") and it need not necessarily be semantically meaningful in the application field. This is different with prototyping data. Since the operator of the prototype is a human being and not an automated test driver, input and results should be meaningful to the tester. Further, the internal properties of a prototype cannot be used to determine the basis for test cases. Therefore, testing should be done along the line of a "black box test". However, with black box texting, test data is derived from a specification. Since this specification is lacking in a pure prototyping methodology (in practical cases it will exist in an informal or rudimentary form only), severe data problems might exist.

In addition to the test-related features mentioned above, genuine prototyping support features will be:

- modification support;
  - support to incorporate changes requested by the experimental subjects (users) as rapid as possible,
  - support composition of changes,
  - documentation support / help feature;
- evaluation support;
  - keep the feedback obtained from the experimental subjects,
  - analyze the feedback contained in the prototyping data base;
- support the transformation of the prototype into the final product.

The feature for modification support is a key feature if prototyping is supposed to be "rapid". However, one should note that not all environments or languages offered for the rapid creation of code lend themselves for easy and rapid modification of code. Therefore, modification support calls for modularity and, if the prototype is to become large, for an indexing scheme to take advantage of this modularity.

The support for composition of changes calls for a system which allows the user to express the desire for a system in which function A is performed as in version 3, function B as in version 5 and function C as in the current version of the prototype, but all other functions should be as in the second but last version of the prototype. Without special support, the prototype developer might consider the fulfillment of this request a formidable task. In a software base, as e.g. proposed in [18] composition of a system according to such a menue selection could be easily done.

The support of a help facility is needed, if the prototype is not only a vehicle for demonstrations but a tool to give the user some hands-on experience of the system he will eventually obtain. Support of a help facility should be part of the environment and not part of the product, since help facilities are in most cases considered unimportant for prototyping purposes. During prototyping, their major aim is to relate to the user which changes have been made with respect to the previous versions of the prototype he has been working with.

It might be noted that the need for such a facility indicates that in spite of the rapidness and inexpensiveness of prototype creation or prototype modification, caution should be exercised with regard to the number of versions a user is confronted with. Though the purpose of prototyping is experimenting, the user as experimental subject must never perceive himself as the guineapig of the experimenter. To avoid this perception, prototyping should be based on a rudimentary formal requirements analysis and the user should be informed about the experimental design of the prototyping phase and obtain the impression that the proto

types he is working with converge to the final product (which is not to say they have to be evolvable).

The evaluation support mentioned as another main feature genuine to a prototyping environment is clearly distinct from the related features of a testing environment. There, the evaluation support helps to find the fault which lead to an error and possibly aids in correcting this fault. Here, evaluation support relates to keeping and analyzing the feedback one gets from users after operating with a particular version of a prototype.

The special problems in this area are, that these feedbacks need not be consistent - a problem familiar to information gathered in classical requirements analysis. Therefore, the analysis features of the prototyping data base should help to contrast or aggregate statements obtained from different persons or with different versions of a prototype. The facilities needed here resemble those used for requirements verification and for arbitration of differences in environments supporting formal requirements analysis.

Support for transformation of the prototype into a final product is addressed to in the next section.

ENVIRONMENTS FOR EVOLUTIONARY PROTOTYPING

Since the advent of the word "prototyping" in connection with software engineering, end-user languages are praised as "prototyping language" and end-user tools are refered to as "prototyping tools". (For an overview see [12].)

Such claims are only partly correct. They are incorrect in so far, as an ad-hoc query to a data base or a once-and-for-all run of some forecasting model cannot be considered a prototype. They lack the teleology of being "an original on which a thing is modeled" (Websters Dictionary). On the other hand, end-user languages have a very high language level in the application domain for which they are destined. Therefore, they are suitable for rapid development with low development cost, possibly bought by high operating cost. This makes them seductive for application to prototyping (e.g. [25]), even if the features for a prototyping environment mentioned in the previous section are lacking and even if there is no support for evolving a rapidly developed prototype into a fast running system.

As examples for supporting a gradual transition from a prototype to an operating product I will refere to the languages HIBOL [16b] and SETL [3]. Both languages have a very high level for their domain of primary application (business applications for HIBOL, problems lending themselves for set-theoretic formulations for SETL), and both provide facilities for gradually tuning a program by selectively reducing the language level or refining the respective program.

In HIBOL, programming is done in various phases, the last of which is an optional optimization phase. In this phase, the programmer can selectively tune data structures. He can do so by indicating the usage pattern or (/and) the volume of this data structure and the system will decide on a suitable implementation based on this information. Alternatively, he might suggest a particular implementation strategy [14].

This selective tuning is made possible because the HIBOL translator generates for each repetitive data structure a data capsule in the target language of the translator [10]. These capsules can be considered as instantiations of a data abstraction in which the implementation part can be replaced without affecting the interface specification. In extreme cases, the user might even write his own implementation part and have it plugged into the system-generated target code.

SETL relies also on data encapsulation for improving the implementation of the data structures used in a program. It goes even a step further than HIBOL. An optimizer can automatically determine the most efficient data structure from the code of the program [21], [7]. However, this optimization is limited to finding the most efficient representation for the universes (bases) of sets and their efficient implementation in a hash table. There is no support for other, notably manually implemented data structures.

Another interesting feature of SETL is that it allows refinements in the algorithmic part of a program. These refinements are done within the language itself [2]. Thus, one can transform a very high level SETL program into an intermediate level SETL program. The technique applied for such disciplined and provable transformations are basically loop jaming and finite differencing (formal differentiation) [22], [19]. An example of the application of program transformation within SETL is given in [4].

Both, SETL and HIBOL, can be considered as alternative approaches for the gradual evolution of a prototype into an operational product. This can be achieved by selectively adding information to a program. This information can either be used by the heuristics of the system or it might serve to override the decisions made by implementation heuristics.

SUMMARY

The paper focussed on the experimental nature of requirements analysis based on prototyping as compared to the analytical nature of formal requirements analysis. Based on these differences, differences in the requirements engineering environments for the paper and pencile type formal analysis and for a prototyping approach are identified.

Due to the similarity between prototyping and testing, features of a prototyping environment encompass certain features of testing environments. However, genuine prototyping features for supporting easy modification of prototypes and for evaluating prototyping experiments are needed. Since prototyping cannot be done in a stand-alone mode but must be based on preliminary formal requirements analysis, these genuine features should also preliminary help to relate formally expressed requirements to results of the prototyping experiment.

REFERENCES

[1] B. W. BOEHM: "Software engineering economics", Prentice Hall, Englewood Cliffs, N.Y. 1981.

[2] R. B. K. DEWAR, A. GRAND, S.-C. LIU, J. T. SCHWARTZ, and E. SCHONBERG: "Programming by refinement, as exemplified by the SETL representation sublanguage", ACM Trans. on Programming Languages and Systems, Vol. 1/1, July 1979, pp. 27 - 49.

[3] R. B. K. DEWAR, E. SCHONBERG, and J. T. SCHWARTZ: "Higher Level Programming: Introduction to the Use of the Set-Theoretic Programming Language SETL", Courant Inst. of Mathematical Sciences, New York University, New York, 1981.

[4] R. B. K. DEWAR, M. SHARIR, and E. WEIXELBAUM: "Transformational Derivation of a Garbage Collection Algorithm", ACM Trans. on Programming Languages and Systems, Vol. 4/4, Oct. 1982, pp. 650 - 667.

[5] Department of Defense: "Software Technology for Adaptable Reliable Systems (STARS) - Program Strategy", ACM Software Engineering Notes, Vol. 8/2, Apr. 1983, pp. 56 - 108.

[6] J. C. ENOS and R. L. van TILBURG: "Software Design", In: R. W. JENSEN and C. C. TONIES (eds.) "Software Engineering", Prentice Hall Inc., Englewood Cliffs, N. J. 1979.

[7] S. M. FREUDENBERGER, J. T. SCHWARTZ and M. SHARIR: "Experience with the SETL Optimizer", ACM Trans. on Programming Languages and Systems, Vol. 5/1, Jan. 1983, pp. 26 - 45.

[8] J. HOOPER and P. HSIA: "Scenario-Based Prototyping for Requirements Identification", ACM Software Engineering Notes, Vol. 7/5, Dec. 1982, pp. 88 - 93.

[9] P. HSIA and R. T. MITTERMEIR: "Conceptual foundations for a scenario driven approach to assess system requirements", working paper, U-Arlington, UMCP and TU-Wien, Dec. 1982.

[10] M. KNAPP: "Wartungshandbuch f. HIBTRANS - Version 2" TR DA 83/05/02, Institut f. Angewandte Informatik und Systemanalyse, Techn. Univ. Wien, Mai 1983.

[11] M. M. LEHMAN: "Program Evolution, Programming Processes, Programming Support", Sonderdruck zu: W. SAMMER, H. MORGENBROD "Programmierumgebung und Compiler", Berichte German Chapter of the ACM, 18a, B. G. Teubner, Stuttgart 1984.

[12] J. MARTIN: "Application Development Without Programmers", Prentice-Hall Inc., Englewood Cliffs, N.J. 1982.

[13] R. T. MITTERMEIR, G. AICHHOLZER, G. WALLER: "International Comparative Study on the Organization and Performance of Research Units - The Austrian Feedback-Seminares: A Critical Review": Institut fuer Hoehere Studien und Wissenschaftliche Forschung, Wien, Oct. 1977.

[14] R. T. MITTERMEIR: "HIBOL - A very High Level Business Oriented Language, User Manual", TR DA 81/04/04, Institut fuer Angewandte Informatik und Systemanalyse, Technische Universitaet Wien, Wien, 1981.

[15] R. T. MITTERMEIR, P. HSIA and R. T. YEH: "Alternatives to overcome the communications problem of formal requirements analysis", Proc. International Symposium on Current Issues of Requirements Engineering Environments, Kyoto, Japan, Sept. 1982, pp. 163 - 169.

[16] R. T. MITTERMEIR: "HIBOL, A Language for Fast Prototyping in Data Processing Environments", ACM Software Engineering Notes, Vol. 7/5, Dec. 1982, pp. 133 - 140.

[17] R. T. MITTERMEIR, N. ROUSSOPOULOS and R. T. YEH: "Objectives Analysis", Proc. 16th Annual Hawaii International Conference on System Sciences, Honolulu, Hawaii, Jan. 1983.

[18] R. T. MITTERMEIR: "Software Bases for adaptive Maintenance of complex Software Systems", in: 7. Int. Kongress Datenverarbeitung im Europaeischen Raum, "Informationstechnologie: Realitaet und Vision" 19. - 23. Maerz 1984, Wien, ADV, 1984, pp. 483 - 492.

[19] R. PAIGE and S. KOENIG: "Finite Differencing of Computable Expressions", ACM Trans. on Programming Languages and Systems, Vol. 4/3, July 1982, pp. 402 - 454.

[20] R. L. PATTON: "Prototyping -- A Nomenclature Problem", ACM Software Engineering Notes, Vol. 8/2, Apr. 1983, pp. 14 - 16.

[21] E. SCHONBERG, J. T. SCHWARTZ, and M. SHARIR: "An Automatic Technique for Selection of Data Representation in SETL Programs", ACM Trans. on Programming Languages and Systems, Vol. 3/2, Apr. 1981, pp. 126 - 143.

[22] M. SHARIR: "Some Observations Concerning formal Differentiation of Set Theoretic Expressions", ACM Trans. on Programming Languages and Systems, Vol. 4/2, Apr. 1982, pp. 196 - 225.

[23] S. L. SQUIRES, M. BRANSTAD, and M. V. ZELKOWITZ: "Special Issue on Rapid Prototyping", ACM Software Engineering Notes, Vol. 7/5, Dec. 1982.

[24] W. SWARTOUT and R. BALZER: "On the Inevitable Intertwining of Specification and Implementation", Comm. ACM, Vol. 25/7, July 1982, pp. 438 - 440.

[25] P. TAVOLATO and F. VINCENA: "A Prototyping Methodology and its Tool", Proc. Working Conference on Prototyping, Oct. 1983, Namur, Belgium.

[26] P. ZAVE and R. T. YEH: "Executable Requirements for Embedded Systems", Proc. 5th Int. Conf. on Software Engineering, San Diego, 1981, pp. 295 - 304.

A FORMAL APPROACH TO THE PROBLEMS OF MICROCODE COMPACTION CAUSED BY TRANSITORY DATA RESOURCES OF THE MICROARCHITECTURES

L. Manasiev, A. Petkov, K. Boyanov
BULGARIAN ACADEMY OF SCIENCES
1090 SOFIA

UDK: 681.3.02

ABSTRACT

A method of sequential microcode compaction in architectures, containing transitory data resources is discussed in the given paper.

A theoretical apparatus, allowing to find out those microoperation groups requiring synchronization according to their transitory data resources as well as their corresponding transitory data resources is suggested.

An algorithm, separating the microoperation groups resulting from the transitory data resources in the microarchitectures is discussed too.

1. INTRODUCTION

The horizontal microprogramming is widely applied in the nowadays computer systems design. One of the most perspective approaches to horizontal microprograms creation is to describe the control algorithms as a sequence of statements of a general purpose, architecturally independent algorithmic language. The described algorithms are translated into an equivalent sequence of microoperations - sequential microcode. However, no possibilities for concurrent execution of some of the microoperations are reflected in it. This does not allow effective usage of the horizontal microinstruction potential for concurrent execution of several microoperations. Thus, it naturally leads to the problem of sequential microcode conversion into an equivalent concurrent microcode. It is known under the name of "microcode compaction" [1].

The used compaction method in [2], [3], [4], et al. fixes the resource, time and format dependences between the different microoperations. This allows creation of methods for automatic determination of the possible parallelism in the sequential microcode.

All mentioned investigations are characterized by the assumption that the resources preserve intact their state until it is changed by some microoperation. However, there are resources in the microarchitectures which could preserve their contents only for a definite period of time. These resources are known as "transitory data resources". The transitory data resources problem is well stated in [5], but no solution is provided.

A method for sequential microcode compaction in architectures containing transitory data resources is presented in the given paper.

2. PROBLEM FORMULATION

Let's assume that $M = \{MOP_1, \ldots, MOP_i, \ldots, MOP_j, \ldots, MOP_k\}$ is the sequential source microcode. Let's assume also that MOP_i microoperation updates the contents of the transitory data resource T, and that the correct MOP_j execution requires the contents T defined by MOP_i. Thus, MOP_i and MOP_j are strongly limitted in terms of their participation in the compaction process. Hence, their allocation in the microinstructions in the resultant concurrent microcode should provide the defined contents T during MOP_j activation.

Two independent stages could be defined in the solution of the so formulated problem:

I. Determination of the microoperation groups in M, requiring synchronization according to the transitory data resources as well as of the corresponding transitory data resources too.

II. Execution of the compaction process. Then, the sequential source microcode is modified as $M' = \{MB_1, \ldots, MB_s\}$, where MB is the set of microoperations, participating in a synchronization group.

The well developed methods in [2], [3] could be applied to the execution of the compaction process over the microcode M'.

3. FORMAL DESCRIPTION OF THE DEPENDENCES \CC ACCORDING TO TRANSITORY DATA RESOURCES

Let's assume that MOP_i, MOP_j are elements of the sequential source microcode M, where $i < j$, i.e. MOP_i precedes MOP_j in M.

DEFINITION 1:

"Dtd" (data transitory dependence) relation is set between MOP_i and MOP_j, if the following conditions are satisfied for at least one transitory data resource T:

1. The resource T is updated by MOP_i.

2. The available contents T before the updating of MOP_j is required for the correct MOP_j execution.

3. There is no MOP_k element from M, i.e. $i < k < j$ and MOP_k updates the contents T.

Let's assume that there are 'n' transitory data resources in the microarchitecture the M microcode is destined for. Without any limitation, they could be numbered and a sequentially numbered set must be generated $\{T_1,\dots,T_n\}$. On this basis, two vectors can be connected with each M element: TWV (transitory write vector) and TRV (transitory read vector):

- $TWV = (x_1,\dots,x_n)$, where
$$x_i = \begin{cases} 1\text{(true)}, \text{ if } T_i \text{ satisfies condition 1} \\ \text{from Def. 1.} \\ 0\text{(false)}, \text{ otherwise.} \end{cases}$$

- $TRV = (y_1,\dots,y_n)$, where
$$y_i = \begin{cases} 1\text{(true)}, \text{ if } T_i \text{ satisfies condition 2} \\ \text{from Def. 1.} \\ 0\text{(false)}, \text{ otherwise.} \end{cases}$$

The couple (TWV,TRV) which is compared with every microoperation from M represents its potential according to dtd relation. It could be used for strict determination of the place of a M element in the existing M chains of dtd-dependences. The following classification is used for its definition:

a. $M1 = \{ MOP_i/MOP_i \text{ M and } TWV_i = \bar{0},\ TRV_i \neq \bar{0}$

b. $M2 = MOP_i/MOP_i \text{ M and } TWV_i \neq \bar{0},\ TRV_i = \bar{0}$

c. $M3 = MOP_i/MOP_i \text{ M and } TWV_i = \bar{0},\ TRV_i = \bar{0}$

d. $M4 = MOP_i/MOP_i \text{ M and } TWV_i \neq \bar{0},\ TRV_i \neq \bar{0}$

It is evident that $M = \bigcup_{i=1} M_i$ and $M_i \cap M_j = 0$ for $i,j = 1\div4$, $i \neq j$. The elements of the different classification classes have the following logical meaning:

- $MOP_i \in M1.MOP_i$ can only end a chain of dtd-dependences in M, but it cannot be the first or middle member of the chain. This is due to the fact that it is not possible to exist $MOP_j \in M$, $i \neq j$ and MOP_i dtd MOP_j.

- $MOP_i \in M2.MOP_i$ can lead to the formation of a chain of dtd-dependences but it is possible for it to be only the first member of the chain.

- $MOP_i \in M3.MOP_i$ cannot be a member of the chain of dtd-dependences.

- $MOP_i \in M4$. Since this is the general case, nothing can be said about these elements.

In terms of the further reasonings, the following WT (weight) function is defined for the TWV and TRV vectors in the following way:

$$WT(TWV) = \sum_{i=1}^{n} x_i \quad ; \quad WT(TRV) = \sum_{i=1}^{n} y_i$$

The function $WT(TWV_i)$ gives us the number of these transitory data resources, which contents is updated by MOP_i, while the function $WT(TRV_i)$ gives us the number of the transitory data resources which contents is required for MOP_i execution.

3.1. THEOREMS CONCERNING dtd-RELATION

The following theorem was formulated and proved which provides valuable criteria for the existence dtd relation between two elements from M.

T1:

Let MOP_i, $MOP_j \in M$ and $i < j$, then MOP_i dtd if the following conditions are satisfied:

a. $\bar{Z} = TWV_i \ \&\ TRV_j \neq \bar{0}$

b. $WT(\bigvee_{k=i+1}^{j-1} TWV_k) \ \&\ \bar{Z}) < WT(\bar{Z})$

It expresses the connection between the entered relation dtd and the characteristic vectors.

The function $LDTD(MOP_i,MOP_j)$ is defined in order to express quantitatively the degree of complexity of the corresponding microoperations at dtd relation. It specifies the number of those transitory data resources which cause the appearance of a dtd relation between MOP_i and MOP_j. The defined in that way function is specified as a "degree of dtd relation". The relationship between the vectors and the defined function follows from the following theorem:

T2:

If MOP_i dtd MOP_j, then $LDTD(MOP_i,MOP_j) = WT(\bar{Z}) - WT(\bigvee_{k=j+1}^{j-1} TWV_k)\&\bar{Z}) = WT(\bar{Z}\ \&\ \bigvee_{k=j+1}^{j-1} TWV_k)\&\bar{Z}))$

4. dtd ANALYSIS

On the basis of the developed theories, it is

possible to perform analysis in the M set in order to establish the dtd relations, existing between its elements. This process is called dtd analysis. Its objective is to build the following oriented loaded graph:

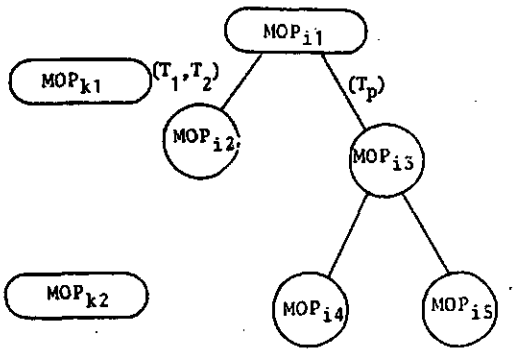
- vertexes: M elements

- edges: there is an edge $\overline{MOP_i, MOP_j}$ if between the corresponding vertexes $MOP_i dtd MOP_j$.

- Load: a chain of natural numbers is assigned to the edge $\overline{MOP_i, MOP_j}$ containing the indices of those transitory data resources which cause the appearance of dtd relation between MOP_i and MOP_j. Evidently their number is LDTD (MOP_i, MOP_j).

4.1. dtd GRAPH

The graph, described in the previous section is defined as dtd graph. An example of a dtd graph built on the basis of $M = \left\{MOP_{is}\right\}_{s=1}^{5} \cup \left\{MOP_{ks}\right\}_{s=1}^{3} \cup \left\{MOP_{js}\right\}_{s=1}^{4}$ is given in fig. 1:



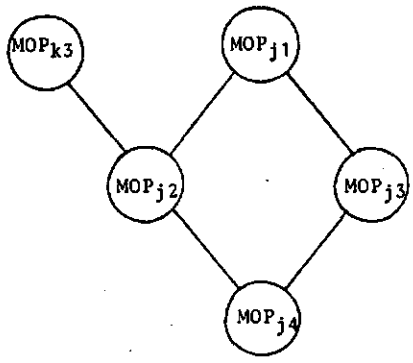


Fig. 1

Generally, the dtd graph will contain single vertexes and subgraphs, which will be called "essential". The separate vertexes, according to their place in the dtd graph will be as follows:

- "single" - e.g.: MOP_{k1}, MOP_{k2}
- "roots" of the essential subgraphs - e.g.: MOP_{k3}, MOP_{j1}
- "leaves" in the essential subgraph - e.g.: MOP_{i4}, MOP_{j4}
- "intermediate" in the essential subgraphs - e.g.: MOP_{i3}.

4.2. THEORETICAL BASIS OF THE dtd ANALYSIS ALGORITHM

An axiomatic condition is stated about the usage of the transitory data resources in order a correct dtd analysis algorithm to be provided. It does not limit the logic of the transitory data resources application in microprogramming and it can be defined as follows: it is assumed that the usage of the available transitory data resources contents by a microoperation is an action which the programmer has consciously accepted, while transitory data resource updating could take place unexpectedly. Hence, it follows that for MOP_i microoperation, which uses the information of a given transitory data resource during its execution, there should be another MOP_i microoperation, i.e. that MOP_i dtd MOP_j in relation to that resource.

The following assertions about the relation between the stated classification classes and the dtd graph have been proved:

Ass.1: The roots of the essential subgraphs are elements of the M2 class.

Ass.2: The single vertexes in the dtd graph are elements of M2 M3 and all M3 elements are single vertexes.

Ass.3: The M1 elements are leaves in the essential subgraphs of the dtd graph.

The following characteristics of M1, M2, M3 and M4 classes could be defined on the basis of Ass. 1, 2 and 3:

- M1 - leaves in the essential subgraphs of the dtd graph
- M2 - roots of the essential subgraphs or single vertexes
- M3 - single vertexes in the dtd graph
- M4 - intermediate elements or leaves in the essential subgraphs of the dtd graph.

The following assertion, allowing decreasing the algorithm steps is very important for the development of an efficient dtd analysis algorithm.

Ass.4: If MOP_i dtd MOP_j according to the transitory resources $T^i = \left\{T_{i1}, \ldots, T_{is}\right\}$, then

for any other MOP_p, i.e. $p \neq i$, $p < i$ and MOP_p dtd MOP_j according to resources $T^p = \{T_{p1},\ldots,T_{ps}\}$, follows $T^j \cap T^p = 0$.

4.3. dtd ANALYSIS ALGORITHM

On the basis of the above assertions the following iterative dtd analysis algorithm is defined:

Step A. A characterizing vector couple (TWV, TRV) is created for every M element.

Step B. M1, M2, M3 and M4 classes are defined.

Step C. M1$\bigcup$M4 elements are ordered in a descending order of their indices.

Step D. The assignment P:=M2 is performed for the working set P.

Step E. The P elements are ordered in a ascending order of their indices.

Step F. The P elements are sequentially processed as follows: Let MOP_a be a consecutive element. MOP_a is checked according to T1 criterion for the existence of dtd relation with all MOP_j, i.e. $MOP_j \in$ M1$\bigcup$M4 and $a < j$. When such a relation is determined, in the corresponding TRV_j vector all components which comform to the transitory data resources causing the dtd relation are set to zero.

Step G. Has P been exhausted? If "NO",go to step F.

Step H. The set $Q = \{MOP_c / MOP_c$ M4 and MOP_a P is formed, i.e. MOP_a dtd $MOP_c\}$. If, $Q \neq 0$, then P:=Q and go to step E.

In the initial step execution, M2 elements, which are roots of essential subgraphs are separated from the single elements. In the next passes, the probable intermediate elements of the essential subgraphs are manipulated with. T2 provides strict information about the loading of the defined dtd connections by means of the non-zero vector components $\bar{r} = \bar{z}$ & $(\bigvee_{k=i+1}^{j-1} TWV_k)$ &$\bar{Z}$) /specially for MOP_i dtd MOP_j/.

5. CONCLUSION

The received in the given paper theoretical results and the developed on their basis dtd analysis algorithm allow to separate the microoperation groups, appearing due to the existence of transitory data resources in the microarchitectures. The elements, defined as vertexes of one and the same essential subgraph must be correctly synchronized which could be performed without any principle dificulties. Further on in the compaction process, the so synchronized elements are defined as an enlarged microoperation. So, the source M set is defined as a M' set of enlarged single elements. The well-known principles of local and global compaction are valid and naturally applied for M' while the existence of the transitory data resources is ignored.

REFERENCES

1. Dasgupta S., Tartar J., The indentification of maximal parallelism in straight-line microprograms, IEEE Trans. on Comp., Vol. C-25, 1976, 10.
2. Tsuchiya M., Gonzales M., Toward optimization of horizontal microprograms, IEEE Trans. on Comp., Vol. C-25, 1976, 10.
3. Tokoro M., Tamura E., Optimization of microprograms, IEEE Trans. on Comp., Vol. C-30, 1981, 7.
4. Mezzalama M., Prinetto P., Microinstruction modeling using timing and semantic constraints, *Simulation, Modeling and Development*, Cairo, 1981.
5. Landskov D., Davidson S., Some experiments in local microcode compaction for horizontal machines, IEEE Trans. on Comp., Vol. C-30, 1981, 7.
6. Boyanov K., Petkov A., Manasiev L., Microprogramming of bit-slice microsystems, internal report of Institute of Mathematics BAS.

MBPL MICROPROCESSOR DEVELOPMENT CROSS-SYSTEM: THE EMULATOR SUBSYSTEM

Boyan Yankov, Lilyan Nikolov, Stoyan Bonev

Higher Institute of Mechanical and Electrical Engineering

Sofia, Bulgaria

UDK: 681.3.06

This paper presents the emulator subsystem of a microprocessor software development cross-system. The basic intent of the designers of the cross-system was to produce a language and its compiler, comparable with the assembly language in universality and efficiency. Another important requirement was a portability of the developed software to different types of 8-bit microprocessors. To achieve this, an intermediate language in a postfix form and a virtual processor are used. The emulator simulates the functions of the virtual processor. It is a convenient tool to debug and to verify programs, written in the input language of the cross-system, called MBPL. The emulation process is controlled by a set of directives. The advantages of the emulation process and a methodology to build such debugging tools are discussed in the paper as well.

1. INTRODUCTION

A possible approach to produce a portable software is to build virtual processors. This paper describes an emulator of such a virtual processor, written especially for an implementation of a microprocessor programming language, called MBPL (Microprocessor Basic Programming Language) [1]. The emulator, described in the paper, is a subsystem of the MBPL microprocessor development cross-system and offers efficient facilities for debugging and verification of MBPL source programs.

The MBPL cross-system includes the next components: 1) the MBPL programming language, oriented to discrete control and system software problems; 2) MBPL compilers; 3) an assembler of a virtual processor; 4) virtual processors; 5) an emulator. The compiler [2] generates an intermediate postfix symbolic instructions code, represented in the form of an assembly language. Next, this intermediate program is translated by the assembler [3] into a virtual processor machine language program. Thus, the machine program is microprocessor independent - the hardware dependence is transferred to the virtual processor. Virtual processors, based on M 6800 and I 8080/85, are available [4, 5]. The user is free to construct his own virtual processor, using any 8-bit microprocessor. The virtual processors are organized as stack machines and interpret an object code in a postfix form. The generated machine program may be executed either in an emulation mode, or it may be punched on a paper or on a magnetic tape. Thus, it is ready for a later use on a microprocessor development system, where it may be EPROM programmed and directly interpreted by a resident virtual processor.

The programming language of MBPL, from the semantic point of view, can be considered as a medium-level programming language. From one side the language statements are typical for the high-level programming languages, and from the other side MBPL offers some features, proper to the machine-oriented languages in sense to deal with data in its internal representation. The following MBPL source program illustrates the language structure:

```
   -- PROGRAM GENERATES ASCII CODES
      OF ALPHA-NUMERIC CHARACTERS     --
  BEGIN Ø1ØØH;
    DECL LETTER(26),DIGIT(1Ø);
    DECL I,CARRY;
  BEGIN 2ØØH;
START: DIGIT(Ø)='Ø';
       LETTER(25) = 'Z';
MID:
       FOR I = Ø BY 1 TO 9
         DO
           DIGIT(I)=INCR DIGIT(DECR I);
         ENDO;
       I = 24;
       WHILE  I ¬= Ø
         DO
           LETTER(I) = DECR LETTER(I+1);
           I = DECR I;
         ENDO;
FIN:   -- END OF PROGRAM --
    END
```

The language syntax is detaily described in [1]. Here is a brief description of the language. Data length - variables and constants, is one byte. Some operations permit the use of multibyte operands too. Constants in their external representation can be unsigned decimal integers, hexadecimal integers and characters. A variable, preceded by a $ sign, is considered as a pointer. Unidimensional arrays can be used as well. The subscript can be an arbitrary expression. The expressions include the following operations in descending priority: INCR, DECR, NOT - increment by 1, decrement by 1, negation; *, /, MOD - multiplication, division, modulus; +, -, addition, subtraction; <, >, <=, >=, =, ¬= - relations; & - logical AND; ! - logical OR; # - exclusive OR; <-, ->, <@, @>, BIT - shift left, shift right, shift left cyclic, shift right cyclic, bit check. The group of declarative statements is used to define variables and arrays (DECL) and to initialize them (DATA, TEXT). The set of executable statements contains an assignment statement, conditional branch (IF ... THEN ... ENDIF), unconditional

branch (GOTO), two versions of a loop statement, repeating a fixed number of times (FOR ... BY ... TO ... DO ... ENDO) and iterating an undefinite number of times (WHILE ... DO ... ENDO), call for and return from MBPL or machine language parameterless subroutines statements (CALL, CALS, RET). The input/output statements (GET, PUT) provide to give an access to physically addressable communication channels. There exist facilities to control an interrupt system (ENABL, DSABL), to form time delay intervals (WAIT). In case of multimicroprocessor systems one can use primitives to control common storage (TAKE, FREE).

An alternative version of MBPL compiler exists. It produces a program in an assembly language for the selected microprocessor.

The translation process can be directed by a set of options. They serve to print or not a source listing, a table of the names used - identifiers and labels, error messages, to generate or not an object program etc.

2. ADVANTAGES OF THE EMULATION

As it was said, the emulator simulates the functions of a virtual processor and it is designed to execute object programs, generated by the assembler of the MBPL system. The emulation mode offers some advantages and allows the user to debug and check MBPL source programs prior to committing them to EPROM firmware. They are:

a) The emulator operates in the same computing medium as the input program compilation and the intermediate program assembly.

b) Real virtual processors do not provide debugging technics to trace and execute user programs. Such features, if they are present, will really increase time consumption during the program execution. The emulation mode is not a real time process and that is why time consumption is not a critical parameter. So, the emulator, but not the virtual processor, provide specific debugging features to trace object programs execution. They are the following: i) to control storage distribution for the MBPL source program; ii) to detect errors due to invalid operation codes or invalid operands of the assembler instructions, due to arithmetic exceptions, such as division by Ø, due to a full or an empty stack of the virtual processor. As well as the assembler, the emulator algorithm is microprocessor independent because it simulates an execution of a machine-independent object program.

3. EMULATOR BUILDING

Building simulating facilities as emulators imposes the following actions to be performed:

a) Simulation of the hardware configuration (registers, stack, memory locations) of the processing unit, on which the emulated program is being really executed.

b) Simulation of the address space of the emulated program - instructions and data.

c) Trace the execution of an object program from a selected entry point - for a definite number of instructions or during a fixed time interval.

d) Modifying and display of selected memory locations contents of the emulated program.

The emulator described processes the object program in one pass. To achieve the requirements mentioned above, the emulation mode is controlled by a set of directives with the following meaning:

1) SR P<address>
It defines an initial value for the program counter of the virtual processor. Default value is Ø.

2) T <address>
TN<address>
It defines an end address for trace of an object program execution and starts a trace procedure. Successive emulated instructions may be printed (T <address>) or not (TN<address>).

3) R <number>
RN<number>
It defines a time interval for trace of an object program execution and starts a trace procedure. Successive emulated instructions may be printed (R <number>) or not (RN<number>).

4) DM<address>,<number>
It defines a memory address to display a definite number of successive memory locations.

5) SM<address>,<value>,<value>,...
It defines a memory address, from which successive memory locations contents is loaded with definite values.

6) SZ<size>
It defines a size for the stack of the virtual processor. Default value is 5Ø.

7) IM
It starts the emulation process.

8) EX
It ends the emulation process.

The directives' operands <address>, <number>, <size>, <value> are defined as valid constants of the MBPL language - decimal or hexadecimal. Beyond the possibilities of other emulators (for example, MOTOROLA 6800 Simulator), the emulator of the virtual processor permits the user to trace an object program for a definite time interval. Also specific entry points of the source program may be defined for the <address> operand not only as decimal or hexadecimal constants, but as a symbol name, label of the input program. This makes free the user not to organize an address correspondence for the selected entry points of the program. To achieve a desired emulation mode, one has to choose the corresponding emulator directives. Fig. 1 shows how directives may be arranged according to the graph scheme. In case of an invalid order or an invalid syntax of a directive, an appropriate error message is to be displayed and the current directive is being ignored.

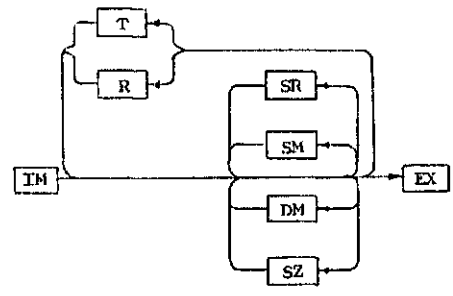


Fig. 1

The emulator algorithm is shown in Fig. 2. An every directive is analysed and branched. The SR directive loads an address for a beginning of a trace procedure in the variable, na-

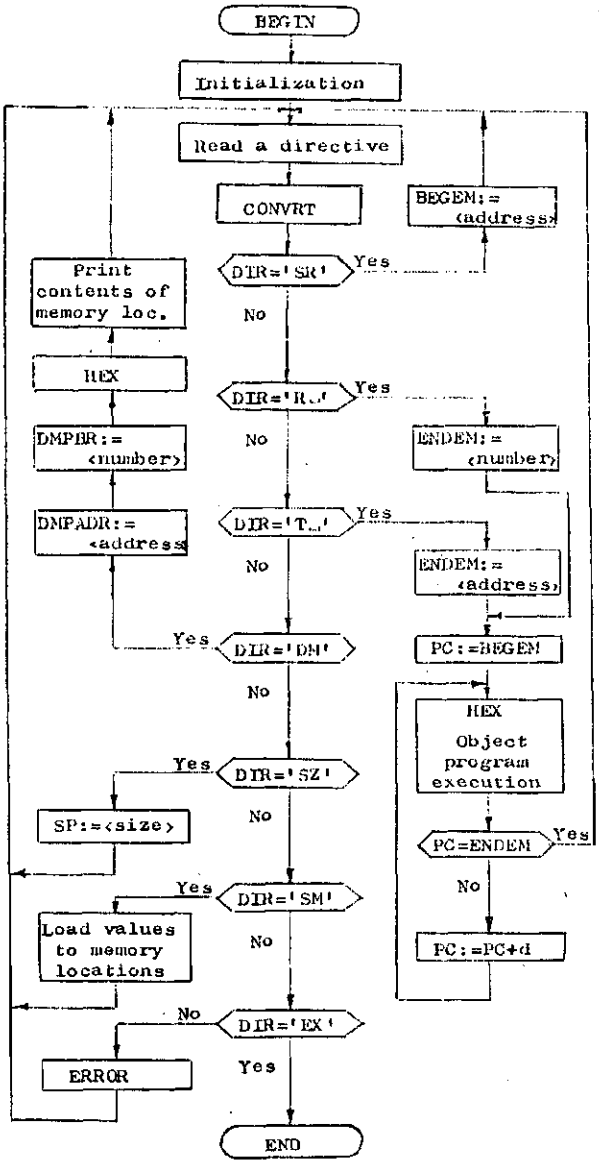


Fig. 2

med BEGEM. The T directive loads the end address for execution in the variable, named ENDEM and starts a trace procedure. The procedure is organized as a loop from BEGEM to ENDEM values. The program counter PC is incremented by d, which varies from 1 to 3 and depends on the object instruction emulated. It is possible to display the following information during the trace: a memory address, an object and mnemonic code of the emulated instruction, a program counter value, the top-of-stack contents, time consumed for object program execution. The R directive action is similar to that of the T directive. The end of trace is defined not as a final address reached, but as a consumed time interval. The DM directive loads a start address and a number of memory locations to the variables, named DMPADR and DMPBR and starts a procedure for display the contents of memory locations. The SM directive loads specific values in definite memory locations and the SZ directive loads a value for the stack size of the virtual processor. The emulator uses the following subroutines: CONVRT - to transform MBPL constants to their internal representation; HEX - to transform MBPL constants from an internal form to a string of hexadecimal digits; ERROR - to display an error message.

The emulator, as well as all the cross-system, is written in FORTRAN IV and ASSEMBLER IBM. It takes about 80K of memory and operates on IBM 360/370 under DOS/OS control.

4. EXECUTION IN AN EMULATION MODE

An example of the emulation process is given in Fig. 3. It shows a protocol of the Section 1 source program emulation. All emulator directives are displayed, preceded by the '>>' signs. They are enclosed in the directive brackets IM for the beginning of an emulation and EX for the end of an emulation. The directive SM LETTER,41H means that the hexadecimal value of 41 is to be assigned to the first element of the LETTER array. DM LETTER,8 means to display the contents of eight successive memory locations, beginning from the address of LETTER. Information is displayed in the following form: the beginning address (Ø1ØØ) and the hexadecimal contents (41 ØØ ØØ ...). The program counter of the virtual processor may be loaded (SR P...) either with a definite MBPL constant (Ø2ØØH) or with the address of a name, used in the source program (MID). T MID and TNFIN define the end address of a trace procedure and start it with (T) or without (TN) a display of the next information, titled as follows: LOC for the address of the emulated instructions, OBJ CODE and MNEMONIC for the object and mnemonic codes of the intermediate language instructions, PC and SP for the program counter and the stack pointer values, TOPSK for the top-of-stack value and TIME for the time, really consumed during the execution of the source program.

```
>>IM
>>SM LETTER,41H
>>DM LETTER,8
Ø1ØØ 41 ØØ ØØ ØØ ØØ ØØ ØØ ØØ
>>SR P2ØØH
>>T MID
 LOC  OBJ CODE  MNEMONIC      PC   SP TOPSK TIME
Ø2ØØ  Ø8 Ø1 1A LADR DIGIT    Ø2Ø3 1C   1A   11Ø
Ø2Ø3  Ø2 ØØ     LDIM ØØ      Ø2Ø5 1B   ØØ   174
Ø2Ø5  4A        INDA         Ø2Ø6 1C   1A   238
Ø2Ø6  Ø2 3Ø     LDIM 3Ø      Ø2Ø8 1B   3Ø   3Ø2
Ø2Ø8  47        STOR         Ø2Ø9 1E   ##   378
Ø2Ø9  Ø8 Ø1 ØØ LADR LETTER  Ø2ØC 1C   ØØ   464
Ø2ØC  Ø2 19     LDIM 19      Ø2ØE 1B   19   528
Ø2ØE  4A        INDA         Ø2ØF 1C   19   592
Ø2ØF  Ø2 5A     LDIM 5A      Ø211 1B   5A   656
Ø211  47        STOR         Ø212 1E   ##   732
  DURATION OF THIS TRACE IS  732 MICROSECONDS.
>>SR PMID
>>TNFIN
  DURATION OF THIS TRACE IS 8918 MICROSECONDS.
>>DM LETTER,1Ø
Ø1ØØ 41 42 43 44 45 46 47 48 49 4A
>>DM DIGIT,1Ø
Ø11A 3Ø 31 32 33 34 35 36 37 38 39
>>EX
```

Fig. 3

The fragment, shown in the figure, traces the MBPL source program from an entry point address Ø2ØØ (labeled START in the source text) to another point, labeled MID. The next intermediate language instructions are emulated: LADR, which pushes to the stack a two-byte address of its operand (DIGIT, LETTER); LDIM, which pushes to the stack a constant (ØØ, 3Ø, 19, 5A); INDA, which modifies an address with a top-of-stack value; STOR, which stores a top-of-stack

value in a memory location, which address is written in the next to top-of-stack element. The run time of the emulated program is given at the end of the trace. The second trace shown is started from a label, named MID and is terminated to a label, named FIN.

5. CONCLUSION

The experience of the MBPL cross-system shows its efficiency to design a system and an applied software for microcomputers and digital devices, based on microprocessors. For example, the cross-system was used to develop a resident MBPL microcomputer system, including a monitor, a text editor and a compiler of the language of MBPL. All these components are written in MBPL, translated and debugged on the cross-system, using the bootstrap principle.

The object code, generated by the MBPL system, is about 1,2 ÷ 1,3 longer, comparing with programs, written directly in the assembly language. Time consumption is increased meaninglessly due to the use of a virtual processor, but this is not essential in most applications.

A new version of the MBPL system, oriented to 16-bit microprocessors, exists and it is similar to the system, presented here.

6. REFERENCES

[1] Yankov B., Nikolov L., A microprocessor programming language and its implementation, MELECON'83, vol. 1, pp A7.05, Athens, Greece.
[2] Yankov B., Nikolov L., A portable compiler for a microprocessor programming language MBPL, Automatics and computer technics, 5, 1983, pp 63-68, Riga, USSR.
[3] Yankov B., Nikolov L., Bonev S., An assembler for a virtual processor, VIII Symposium in Informatics, JAHORINA'84, Sarajevo, Yugoslaviya.
[4] Yankov B., Nikolov L., A virtual processor for microprocessor programming language implementation, ACTA POLYTECHNICA, SPN, vol. 20, III, 5, pp 173-176, Prague, Czechoslovakia.
[5] Yankov B., Nikolov L., An interpreter for INTEL 8080/85, MICROELECTRONICS'82, Siofok, Hungary.

BENCHMARKING MICROCOMPUTERS FOR SOME MATHEMATICAL AND SCIENTIFIC COMPUTATIONS

M.A. Brebner

Department of Computer Science,
University of Calgary,
CALGARY,
Alberta T2N 1N4,
CANADA.

UDK: 681.3.02

Abstract

It is now possible to consider using microcomputers, such as the IBM PC, for the solution of serious scientific problems requiring numerical methods in their solution and, therefore, a large amount of computation. Even the problems which tested the limits of the super computers of 25 years ago can now be attempted on the latest microprocessors. However, the efficiency of the software used can significantly effect the usefulness of these microcomputers for solving serious scientific problems.

The paper will discuss the great range of benchmark times observed using various compilers, semi-compilers and interpreters for BASIC, FORTRAN and PASCAL, on IBM PC Junior, PC, PC XT and PC AT microcomputers. For a subset of the cases results will be compared with other microcomputers such as IBM compatibles and the Apple Macintosh.

If an IBM PC with 64KB memory and at least one 360KB drive is available at the conference a demonstration of some of the benchmarks can be incorporated into the presentation of the paper.

Introduction

It is now possible to consider using microcomputers, such as the IBM PC, for the solution of serious scientific problems requiring numerical methods in their solution and, therefore, a large amount of computation. Even the problems which tested the limits of the super computers of 25 years ago can now be attempted on computers based on the latest microprocessors, such as those for example, from Intel, Motorola, National Semiconductor and Hewlett-Packard.

However, the efficiency of the software used can significantly effect the usefulness of these microcomputers for solving serious scientific problems. Several of the benchmark times given in the paper will illustrate this.

The paper will discuss the great range of benchmark times observed using various compilers, semi-compilers and interpreters for BASIC, FORTRAN and PASCAL, on IBM PC Junior, PC, PC XT and PC AT microcomputers. For a subset of the cases results will be compared with other microcomputers such as IBM compatibles and the Apple Macintosh.

Table 1 gives benchmark times for a piston/shock problem [for details see 1]. Using several different compilers the execution times ranged from 41 minutes 14 seconds to 31.25 seconds on an IBM PC. Also, in the case of the compiler with the slowest reported time, a slight simplification of the problem was required to avoid a breakdown in the computation caused by a flaw in the software implementation of floating point arithmetic. This shows that some compilers for microcomputers are not always very carefully implemented with respect to the numerical aspects of the language. However, in this particular case, with a later version of the offending compiler, which is designed to use the Intel 8087 mathematical coprocessor, no numerical breakdown occurred and the benchmark time was reduced to 104 seconds.

The piston/shock problem was selected as a fairly typical non-trivial scientific computation for the following reasons:

(a) There are 2 independent variables space and time, and the computation is repeated for many time steps.

(b) There are 6 dependent variables at each mesh point, which implies the frequent referencing of 6 one-dimensional arrays.

(c) A considerable amount of computation is performed on these 6 variables at each mesh point, and this is repeated thousands of times during a typical run.

(d) A moving boundary condition is involved, and special methods have to be adopted to smooth out the theoretically discontinuous shock wave.

Unless stated to the contrary, benchmark times discussed are for the piston/shock problem.

IBM BASIC interpreter and compiler

Running the compiled BASIC version on the IBM PC Junior (374secs) and the IBM PC (170secs) shows the PC Junior takes about 2.2 times as much execution time. As the IBM PC Junior, although still available and supported, is now out of production, the paper will now concentrate on the IBM PC and XT. It should be noted that execution times for the PC and XT will be the same as no disc read or writes were used in the benchmark programs.

The paper will now discuss the effect of (i) using real or integer variables for loop counters and array indexing, (ii) graphics output, and (iii) compiling and compiler options. A summary of the relevant data in seconds from table 1 is given below.

Compiled	Real Index	Integer Index	Int. Index No Graphics
No	1639	1484	1378
Yes	223	170	146
Yes/O	-	136	116

Firstly it is not surprising that it is worth the minor effort of using the DEFINT (IMPLIED in FORTRAN) statement to ensure the variables used for indexing are integer. Secondly, there is a modest time penalty for producing graphics output [see 2 for examples], however the improved interaction for the scientist is well worth the fairly small cost. Thirdly, the improvement in execution speed obtained by using the compiler is quite dramatic. Time reduction factors of about 11 to 12 and 8.7 to 9.5 with and without the /O option are shown. However, the compile and link times are now significant (1 to 2 minutes) compared to the virtually instant response to the RUN command in interpreter

BASIC. Also the disk file sizes are larger for the compiled versions of the program as shown below.

BASIC Execution File Sizes (bytes)

Graphics	Interpreter	Compiled	O/option	With 8087
Yes	1797	3072	23424	23040
No	1723	3072	22656	22272

Note the compiled file sizes are in increments of 128

The reason for the improved performance and increased file size with the O/option is that all the required library routines are stored and explicitly linked in the execution file, and no indirect reference to library routines are made during execution. It can be noted here that the IBM Professional FORTRAN, which has the best execution time, produces an execution file of 33394 bytes. Times for machines equipped with mathematical coprocessor will be discussed later, but it should be noted here that utilizing the 8087 allows a small reduction in the execution file size as no software floating point arithmetic routines are required.Finally, although the interpreter version of BASIC is slow, it does have advantages for the interactive development of programs [see 2].

BASIC with 8087 Support

Two versions 1.01 and 3.03 of the Microway 87BASIC compiler were tested. These replace programmed floating point arithmetic routines with 8087 coded routines in the IBM BASIC compiler, and also store real constants in 8087 (IEEE standard) format. With these compilers there is a reasonable gain in performance as is shown below.

Microway 87BASIC vs. BASICA

Version	IBM	Graphics	No Graphics
BASICA	PC	170	146
BASICA/O	PC	136	116
1.01	PC	150	127
3.03	PC	116	101
3.03/O	PC	78	67
3.03/O	AT	60	52

The times for the IBM AT will be discussed later.

Firstly, the 1.01 version without the /O option is seen to be slightly slower than non-8087 BASICA compiled with the /O option. Also the gains using version 3.03 are not as dramatic as 8087 benchmarks which calculate mathematical funtions thousands of times [6]. These latter benchmarks, by their nature, require little memory access through the 8088, and are mainly computing internally in the 8087. However, the piston/shock problem is more typical of computational modelling, frequently accessing main memory, particularly array elements, as well as frequently carrying out arithmetic operations. Memory access times still have a significant effect in this case. Dongarra [5] reports greater gains for linear equation solvers, as discussed later.

PASCAL

The programs have been written to allow solving the piston/shock problem in plane, cylindrical or spherical geometry (ALPHA = 0,1 or 2). However, the required calculation of volumes involves expression of the form:

R(I+1)^ALPHA - R(I)^ALPHA in BASIC or

R(I+1)**ALPHA - R(I)**ALPHA in FORTRAN,

which have to be written in PASCAL as

EXP(LOG(R(I+1))*ALPHA)-EXP(LOG(R(I))*ALPHA) in PASCAL.

This to some extent accounts for the larger benchmark times for PASCAL. To allow a fair comparison of PASCAL, a strictly plane geometry version of the program was written in BASIC and PASCAL, and the results are given in table 2. Although the PASCAL times are still longer, they are now of the same order of magnitude.

IBM AT

A study of parts of table 1 shows the 6MHz Intel 80286 based IBM PC AT to be approximately 3.1 times as fast as the 4.77MHz Intel 8088 based IBM PC when no mathematical coprocessor is used. For example the best BASICA run times are 116 seconds and 38 seconds for the PC and AT respectively. Clearly the faster clock speed of the AT processor does not account for such a large increase. Various other factors such as the internal design of the 80286, the faster access to main memory, the 16 bit bus (as opposed to the 8 bit bus of the 8088), etc., also help produce this significant increase in performance.

However, the results for the AT are less exciting if a 4MHz Intel 80287 mathematical coprocessor is added and compared to the PC with a 4.77MHz Intel 8087 mathematical coprocessor added. For IBM Professional FORTRAN the 8087 PC (31.25secs) is about 15% faster than the 80287 AT (36.3). With Microway 87BASIC runs show the 80627 AT (52secs) approximately 30% faster than the 8087 PC (67secs). It appears that the highly optimized inline floating point code produced by IBM Professional FORTRAN shows the effect of the slower mathematical coprocessor more dramatically. Of course it should be noted the Professional FORTRAN times are the best for both machines. Further compilation times on the AT are about 4 times as fast as the PC. I understand Intel now have 80287 processor with higher clock rates, and with these faster mathematical coprocessor added, microcomputers like the AT could be very powerful tools for solving moderate size scientific problems. Further the strong interactive features of recent microcomputers would help the scientist to easily develop programs and analyse results [see 2].

It can also be noted here that probably the reason for the superior performance of Professional Fortran over other 8087 compilers tested, is that it produces inline 8087 code rather than calling 8087 coded routines. Of course this is more evident on the PC. The Professional FORTRAN also optimizes by, for example, removing redundant calculations, however the program tested already has most of the redundancy removed.

IBM Compatibles

For the tested IBM compatibles running without graphics output or mathematical coprocessors, the times are almost identical. With graphics, the times for the Hyperion and 4.77MHz. Sperry are 140 and 231 seconds respectively, compared to a time of 136 seconds for the IBM PC. The Sperry graphics has double the vertical resolution, and this may account for some of the significant difference in times. However, the Sperry running at 7.16MHz. without graphics takes only 78 seconds, as opposed to the 4.77MHz. IBM PC which takes 116 seconds. The execution files for these tests were compiled on the an IBM PC, and then read from diskette and run on the other types of microcomputer. This was a test of compatibility for both the diskette drives and machines. Further, some of the time discrepancy for graphics output might be explained by code which is inefficient for the particular graphics hardware of other microcomputers.

Microcomputer vs. Super Computer

To compare microcomputers with a super computer, results of some test runs for the piston/shock problem using FORTRAN on a Cyber 205 vector computer with 2 pipes are being carried out. These indicate the CDC Cyber 205 is from 225 to at least 1125 times as fast as the best IBM PC results. The lowest factor is for vectors of length 25, which is unfair on a machine designed to run efficiently on very large vectors. In the case of the higher factor, vectors of length 1000 were used. Some small improvement in the Cyber 205 performance is expected for even larger vectors.

Linear Equation Solvers

Dongarra [5] has produce a report containing a large collection of benchmark times for the solution of a set of linear equations of order 100, using the FORTRAN program for dense linear systems from LINPACK, on a great range of computers. A small relevant subset

of these times in seconds is listed below.

Times from Dongarra Report

Computer	Flt.Pt. Unit	Precision	Time
Apple III	No	Half	2813.0
IBM PC	No	Half	1225.0
IBM PC	Yes	Half	60.0
VAX/780	Yes	Half	4.13
IBM PC	Yes	Double	98.9
IBM AT	Yes	Double	75.1
Cyber205	Yes	Double	0.082

Adding an 8087 to the PC gives an improvement factor of about 20, but the AT with a 80287 only gives a further gain of 25%. Again a factor of greater than 1000 is observed for the Cyber 205 compared to the IBM PC. These times can be significantly reduced if the BLAS(basic linear algebra subroutines) are coded in the appropriate assembler.

Sieves of Eratosthenes and Simple Tests

For the well published benchmark algorithm for the Sieve of Eratosthenes which computes prime numbers [see for example 3] the execution times for IBM BASICA ranged from 183 seconds (3 minutes 3 seconds) using the interpreter version to 1.5 seconds for the best compiler version. It should be noted that Byte magazine [4] published a time of 33 minutes for BASICA on the IBM PC for this benchmark!

The Apple Macintosh does not possess the capability of using a mathematical coprocessor, and therefore some benchmarks for essentially integer arithmetic are given below. These are a subset of those described in Byte [3] for Macintosh BASIC. The tests are,

(a) Sieves of Eratosthenes,

and loops repeated 5000 times which contain,

(b) nothing (empty loop),
(c) GOSUB to an empty subroutine,
(d) substring extraction by A$=MID$(B$,6,7).

	Interpreter BASICA PC	BASICA/O PC	BASICA/O AT	BASIC [3] Macintosh
(a)	182	1.5	0.4	31.5
(b)	4.4	0.11	0.04	1.5
(c)	10.2	0.165	0.06	3.0
(d)	22.0	3.65	1.35	9.0

These figures clearly show interpreter BASIC on the IBM is considerably slower than BASIC on the Macintosh, but compiled BASIC on the IBM is very much faster.

Conclusion

The various tables in this paper give accurate figures for the performance of some microcomputers, and in most cases the tests have been repeated many times, often on different machines of exactly the same type, to ensure the accuracy of the results. In evaluating the execution speed of microcomputers these tables give hard data, rather than the vague personal impressions that are often quoted. Of course there are intangible factors, such as the quality of interaction, but in this area microcomputers are very effective.

ACKNOWLEDGEMENTS

The author would like to thank the following for allowing me to use software available on their machines;
Alan Bays, Geology, John MacRea, Petroleum Engineering, Ann Brebner and Heber Jones, Computer Applications Unit, University Bookstore, University of Calgary. Mark Little, Anil Sahi, Jim Arthurs and Garfield, IBM.

REFERENCES

1. Brebner, M. A., Modelling Fluid Dynamic Shock Wave Problems on a Microcomputer, Society for Computer Simulation 1985 Multiconference, San Diego, Jan. 1985.

2. Brebner, M. A., The Potential of Microcomputer Graphics in the Teaching of Some Classes of Compressible Fluid Flow Problems, Informatica 85, Nova Gorica, Sept., 1985.

3. Byte, April 1984, p 328.

4. Byte, July 1984, p 272.

5. Dongarra, J. J., Performance of Various Computers Using Standard Linear Equations Software in a Fortran Environment, Mathematics and Computer Science Division, Argonne National Laboratory, Technical Memorandum No. 23, 12 Sept. 1984.

6. Dr. Dobb's Journal, March 1984, p92-96.

TABLE 1

Piston/Shock Problem on Microcomputers

This table gives some run times in seconds for a model with 25 meshes and 500 times steps. However, these figures should be taken with some caution as different versions of similar software can produce significantly different times. Also, no attempt has been made to utilize any special programming, such as assembler code, to optimize the execution times.

Computer	Language	Compiler	Graphics	Math Pro.	Run time
IBM PCJr	BASICA	Yes	Yes	No	374
IBM XT	BASICA@	No	Yes	No	1639
IBM XT	BASICA	No	Yes	No	1484
IBM XT	BASICA	No	No	No	1378
IBM XT	BASICA@	Yes	Yes	No	223
IBM XT	BASICA	Yes	Yes	No	170
IBM XT	BASICA	Yes	No	No	146
IBM XT	BASICA/O	Yes	Yes	No	136
IBM XT	BASICA/O	Yes	No	No	116
IBM PC	BASICA MW1.01	Yes	Yes	Yes@1	150
IBM PC	BASICA MW1.01	Yes	No	Yes@1	127
IBM PC	BASICA MW3.03	Yes	Yes	Yes@1	116
IBM PC	BASICA MW3.03	Yes	No	Yes@1	101
IBM PC	BASICA MW3.03/O	Yes	Yes	Yes@1	78
IBM PC	BASICA MW3.03/O	Yes	No	Yes@1	67
IBM PC	FORTRAN PF	Yes	No	Yes	31.25
IBM XT	TURBO V2.0	Yes	No	No	2474**
IBM XT	TURBO-87 V2.0	Yes	No	No	104
IBM AT	BASICA	No	Yes	No	556
IBM AT	BASICA	Yes	Yes	No	56
IBM AT	BASICA	Yes	No	No	48
IBM AT	BASICA/O	Yes	Yes	No	44
IBM AT	BASICA/O	Yes	No	No	38
IBM AT	BASICA MW3.03	Yes	Yes	Yes@1	72
IBM AT	BASICA MW3.03	Yes	No	Yes@1	63
IBM AT	BASICA MW3.03/O	Yes	Yes	Yes@1	60

TABLE 1 (continued)

IBM AT	BASICA MW3.03/O	Yes	No	Yes@1	52
IBM AT	FORTRAN 2.0	Yes	No	Yes	43.5
IBM AT	FORTRAN PF	Yes	No	Yes	36.3
HYPERION	BASICA/O	Yes	Yes	No	140
HYPERION	BASICA/O	Yes	No	No	120
HYPERION	BASICA MW3.03/O	Yes	No	Yes@1	68
HYPERION	TURBO-87 V2.0	Yes	No	Yes	104
SPERRY4.77MHz	BASICA/O	Yes	Yes	No	231
SPERRY4.77MHz	BASICA/O	Yes	No	No	116
SPERRY7.14MHz	BASICA/O	Yes	No	No	78

NOTE
1. BASIC MW refers to versions of MicroWay BASIC compiler.

TURBO(-87) refers to versions of TURBO PASCAL (with 8087 support).

FORTRAN PF refers to versions of IBM Professional FORTRAN.

2. @ Array index and loop counter REAL. That is no variables were declared or set integer.

@1 The maths co-processor was referenced by subroutine library calls, rather than compiled inline code.

/O BASICA compilation with /O option.

** Program slightly modified to avoid a numerical bug in TURBO V2.0. This has been corrected in version 3.0.

TABLE 2

Piston/Shock Problem Program for Plane Geometry

Computer	Language	Compiler	Graphics	Math Pro.	Run time
IBM XT	BASICA@2	Yes	Yes	No	109
IBM XT	BASICA@2	Yes	No	No	90
IBM PC	BASICA MW3.03@2	Yes	Yes	Yes@1	59
IBM PC	BASICA MW3.03@2	Yes	No	Yes@1	48
IBM AT	TURBO V2.0	Yes	No	No	602**
IBM AT	TURBO-87 V2.0	Yes	No	Yes@1	77

OPERACIJSKI PODSISTEM ZA PROGRAMIRANJE

Ernest Kocuvan
Železarna Ravne, Jugoslavija

UDK: 681.3.06

Razvoj programske opreme v splošnem delimo v načrtovanje in programiranje. Za razliko od nevsakdanjega, skoraj inovativnega načrtovanja je programiranje kot praktično uresničevanje načrtane zamisli proces, ki se neprestano ponavlja. Razbremeniti programerja odvečnih, vseh neprestano ponavljivih aktivnosti in nepotrebne izgube energije pri njegovem vsakdanjem delu je bila osnovna težnja sodelavcev računalniškega centra Železarne Ravne.

Plod večletnih skupnih prizadevanj je interaktivni programski paket, ki skoraj v celoti avtomatizira tisti del programiranja, kjer pride do rokovanja z operacijskim sistemom in dopušča programerju, da se popolnoma posveti aplikativnim problemom pri programiranju. Vsebino pričujočega referata sestavlja celovita problematika o tej v bistvu preprosti, toda kolikor je znano, pri nas še nikjer v praksi vpeljani zamisli.

In general planning and programming are parts of the software developement. For a difference from unusual, almost inovative planning, the programming as practical realization of planned idea is a proccess that continues interruptedly. The basic tendency of all participant in the computer centre of Železarna Ravne was to discharge the programmer of all unneeded activities and not necessary energy losts at his every day work.

The final result of many years lasting common efforts is an interactive programming package, which automizes almost completely that programming part, where it comes in touch with operating system and enables the programmer to pay his complete attention to his aplication problems. This paper presents the whole problematics of this basically simple idea, but which as known so far has not been introduced yet.

OPERATING SUBSYSTEM FOR PROGRAMMING

1. UVOD

Osnovno in vsakdanje opravilo programerja je programiranje. To je obsežna množica raznovrstnih nalog, ki jih je potrebno opraviti, da pretvorimo številne ideje in zamisli, zapisane v programski definiciji, v uporabno obliko, ki jo na koncu predstavlja pravilno delujoč program.

Obsežni pojem programiranja lažje pojasnimo z grobo razdelitvijo opravil na manjše logično zaključene celote:

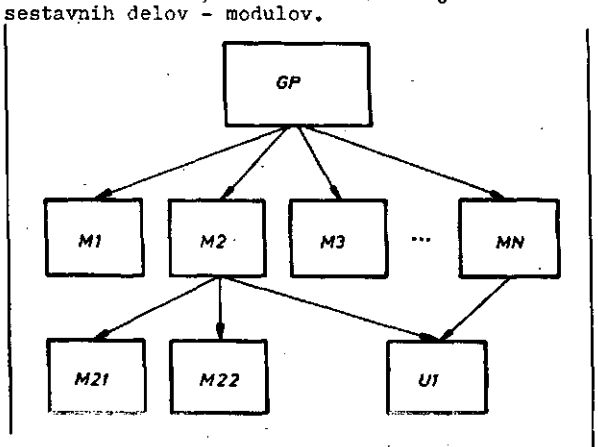
- testiranje programa
- dokumentiranje programa
- arhiviranje programa
- spreminjanje programa

ki jih imenujemo faze programiranja. Pri razvijanju vsakega programa se bolj ali manj intenzivno soočamo z vsako naštetih faz.

1.1 Modularno programiranje

Način pisanja in razvijanja programov se od programerja do programerja zelo razlikuje, toda zaradi številnih in koristnih prednosti se v praksi najbolj uveljavlja princip modularnega programiranja.

Modularno programiranje omogoča hitrejše in enostavnejše pisanje najrazličnejših oblik programa. V osnovi uveljavljene oblike: nemudolarni program, modularni program in programski paket se od aplikacije do aplikacije med seboj lahko zelo razlikujejo tako po razsežnosti kot po raznovrstnosti njihovih sestavnih delov - modulov.



Slika 1.
Fig 1.

GP - glavni program
M_i - programski modul
U_i - univerzalni modul

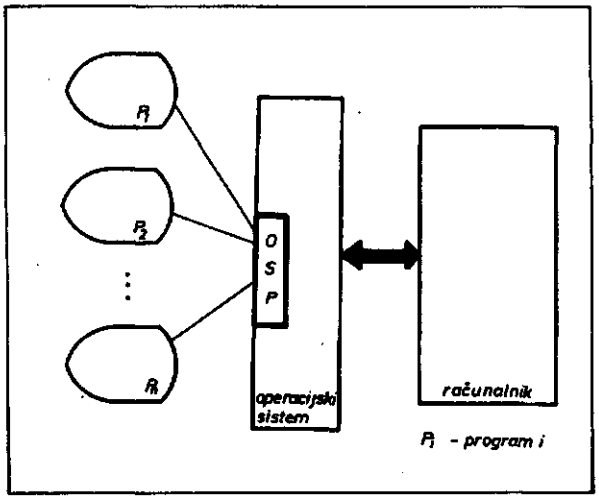
Pri več letni uporabi modularnega programiranja smo opredelili tri osnovne oblike modulov:

- programski moduli
- paketni moduli
- univerzalni moduli

Programski moduli istega programa tvorijo skupno množico izvornih datotek programa. Vsak program ima svojo lastno množico izvornih datotek. V posameznih množicah so odvisno od potreb lahko tudi poljubni splošno uporabni univerzalni moduli. V množicah izvornih datotek programov istega programskega paketa so zraven še za ta paket specifični skupni paketni moduli.

Pri metodi modularnega programiranja običajno napišemo najprej osrednji programski modul (deblo): glavni program, ki mu v skladu s predpisanimi funkcijami programa dodajamo še preostale programske in ostale module (veje drevesa), kar je približno prikazano na sliki 1.

Glavni program se ves čas razvijanja programa spreminja in dopolnjuje. Ostali moduli programa se postopoma dodajajo eden za drugim, v celoti pretestirajo in se običajno do konca pisanja programa več ne spreminjajo, morda le toliko, da jih dokončno uskladimo med seboj.



Slika 2 : mesto operacijskega podsistema za programiranje
Fig.2 : The place of the operating subsystem for programming

1.2 Raznolikost v programiranju

Pri testiranju programa naletimo na dve vrsti opravil, ki se bistveno razlikujeta, sta pa za razvoj vsakega programa nujno potrebni:

A. kodiranje programa
B. priprava programa

Kodiranje programa je v celoti razvojno, inventivno in pri nobenem programu ponovljivo opravilo. Obsega študij programske definicije, uporabo programskega jezika, iskanje slovničnih in logičnih napak, iskanje optimalnih rešitev ipd.

Priprava programa je v primerjavi s kodiranjem rutinsko, neprestano ponavljujoče, monotono opravilo, saj zajema prevajanje modulov programa, povezovanje modulov v sliko programa in končno izvajanje programa. To delo je za programerja popolnoma odveč, ker ga še dodatno nepotrebno utruja.

1.3 Komandne datoteke programa

Delno zmanjšanje nekoristnega napora že omogoči operacijski sistem. Operacijska sistema DELTA-M firme ISKRA - DELTA in RSX-11M firme DIGITAL sta za to poskrbela z možnostjo uporabe komandnih datotek.

Vsebino komandne datoteke sestavlja niz predhodno pripravljenih ukazov operacijskemu sistemu, ki v določenem trenutku opravijo željeno opravilo. Za testiranje programa lahko programer vedno tvori svoje lastne komandne datoteke še pred pričetkom testiranja z običajnim editiranjem, nato jih po potrebi sproti ažurira.

1.4 Pomanjkljivosti

Za nemoteno rokovanje s komandnimi datotekami programa je potrebno poznati sintakso in delovanje sistemskih pomožnih programov: procesorja komandnih datotek in povezovalnika. To povzroča programerjem zaradi občasne rabe veliko težav, za sam razvoj programa pa to znanje ni potrebno.

Upoštevanje pravil modularnega programiranja sam način programiranja bistveno poenostavi in olajša. Ima pa tudi svojo slabo stran. Preglednost nad celotnim programom se zmanjšuje toda ne zaradi obsežnosti in nepreglednosti izvorne kode, ampak zaradi naraščujočega števila datotek programa. S tem narašča težavnost rokovanja z datotekami na operacijskem sistemu sorazmerno z razsežnostjo programa.

V vseh fazah programiranja si sicer pripravimo komandne datoteke, zato da lahko z njihovo pomočjo enostavneje rokujemo z izvornimi datotekami programa, toda komandne datoteke rešijo le del problemov. Ostale probleme, ki dodatno nastajajo pri razvijanju vsakega programa: posamezno datoteko po pomoti zbrišemo ali pokvarimo, arhiviranje in dearhiviranje vedno večjega števila programov s številnimi datotekami je potrebno dodatno rešiti.

Pri premagovanju naštetih pomanjkljivosti si vsak programer pomaga kakor ve in zna. Te rešitve so le delne od primera do primera različne, neoptimalne in večinoma neenotne. Težave nastopijo pri spreminjanju programa zlasti, ko je preteklo več časa od zadnjih sprememb, pri skupinskem programiranju in pri prevzemanju dela drugega programerja.

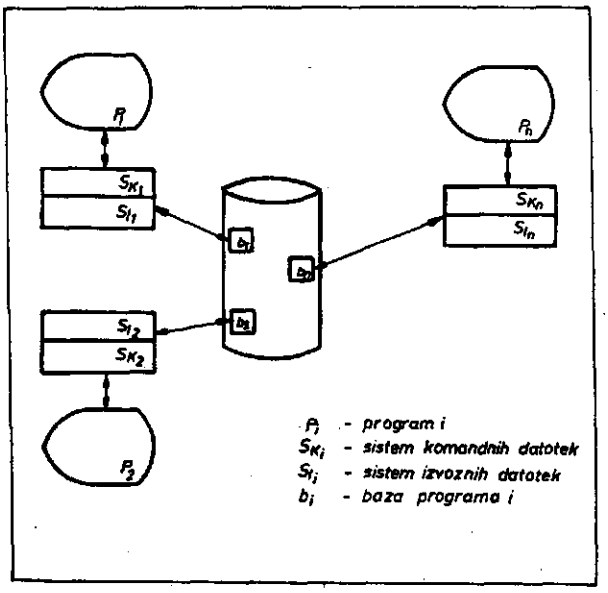
2. REŠEVANJE PROBLEMOV

Da bi se programer lahko popolnoma posvetil razvoju programske opreme in bi ne imel dodatnih težav z operacijskim sistemom, je bil izdelan Operacijski podsistem za programiranje (kratko OSP). Shematični prikaz vidimo na sliki 2.

OSP je vmesnik med programerjem in operacijskima sistemoma DELTA-M in RSX-11M. Njegova naloga je v tem, da za vsak razvijajoči program v programskem jeziku FORTRAN IV ali FORTRAN 77 povsem nadomesti kompletno množico posegov na operacijskem sistemu.

OSP je aktivno prisoten ves čas razvijanja programov na operacijskem sistemu. V tem času tekoče zaznava vse nastale spremembe. Za vsak program obstaja v OSP dvoje ločenih sistemov za rokovanje z datotekami in skupna baza podatkov (slika 3). Vsak sistem zase rokuje s svojimi datotekami pripadajočega programa, tako da jih sproti tvori, dopolnjuje, spreminja in briše.

Vse spremembe se avtomatsko shranjujejo v pripadajoči bazi podatkov. Vsebina in razsežnost je direktno odvisna od razsežnosti programa.



Slika 3: Sestavni deli OSP
Fig 3: Parts of OSP

2.1 Sistem komandnih datotek

Problem rokovanja s komandnimi datotekami programa je bil rešen z uvedbo samostojnega sistema komandnih datotek. Z njegovo pomočjo dosežemo naslednje cilje:
- hitrejša priprava komandnih datotek
- poenostavljeno spreminjanje komandnih datotek
- lažji prehod med opravili vsake faze
- povečanje preglednosti nad celotnim programom
- avtomatsko upoštevanje vseh opcij povezovanja
- enostavno arhiviranje programa

V vsaki fazi programiranja potrebujemo za razvijanje programa komandne datoteke. Te se tvorijo na začetku enkrat za vselej, njihova vsebina se shrani v bazi podatkov, kasneje pa se samo še ažurirajo. Slika 4 prikazuje vse vrste rokovanja s komandnimi datotekami.

V spominu računalnika se avtomatsko pojavljajo, spreminjajo in brišejo odvisno od programerjevih potreb naslednje komandne datoteke:

- glavna (komandna) datoteka
- povezovalna (komandna) datoteka
- segmentna (komandna) datoteka

Vsaka od njih se uporablja za določeno nalogo in ima zato tudi svojo posebno vsebino. Vsebino vsake od njih je možno določiti deloma avtomatsko in deloma interaktivno s pomočjo sistema komandnih datotek (odslej kar OSP).

- glavna datoteka

Vsebino glavne komandne datoteke je sestavljena tako, da vodi celotno testiranje programa. Pripravi jo OSP na več načinov odvisno od faze programiranja.

Pred pričetkom testiranja je novemu programu namenjen spominski prostor še prazen. OSP to ugotovi in avtomatsko tvori začetno vsebino glavne datoteke, ki že omogoča avtomatsko rokovanje z datotekami nemodularnega programa.

Sedaj lahko editiramo glavni program in ga pripravimo za avtomatsko testiranje s pomočjo tvorjene glavne datoteke.

Ko se odločimo za tvorbo prvega programskega modula, postane program modularen. To spremembo v programu takoj s pomočjo OSP interaktivno vnesemo v bazo podatkov. Tudi vsako naslednjo spremembo (dodajanje, brisanje ali spreminjanje oznake programskega modula) v programu je potrebno sproti ažurirati v bazi podatkov.

Tako pozna OSP v vsakem trenutku konfiguracijo našega programa ter nam lahko vedno avtomatsko opravi potrebno operacijo ali posreduje željeno informacijo ves čas testiranja programa.

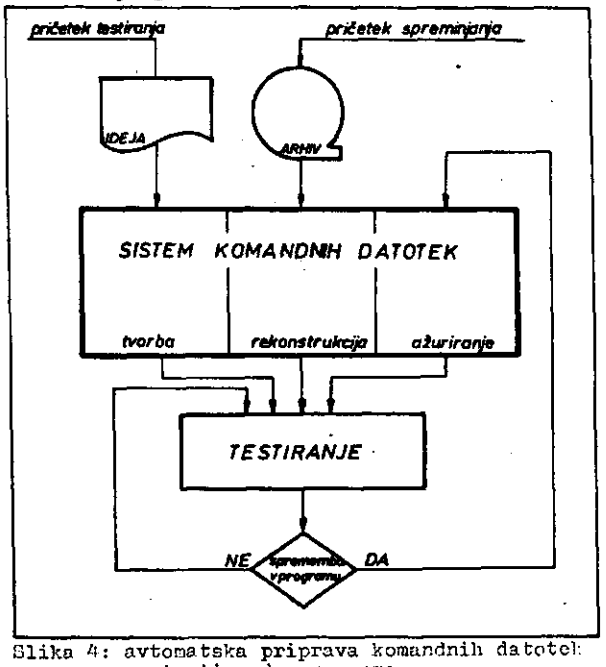
Tudi pred spreminjanjem programa je programu namenjen spominski prostor še prazen. Vanj se iz arhiva skopira arhivska datoteka programa. OSP nam iz nje v trenutku rekonstruira vse potrebne komandne datoteke programa. Sedaj lahko program spet po želji testiramo. Napor in čas prehoda modularnega programa iz mirujoče arhivske oblike v dinamično testno obliko postaneta zanemarljiva.

- povezovalna datoteka

Med testiranjem programa se vsi njegovi moduli povezujejo med seboj v sliko programa. Pri tem je potrebno upoštevati:

- način povezovanja
- optimalnost povezovanja

Trenutna verzija OSP nam omogoča nesegmentno in segmentno povezovanje modulov programa. Pri tem je možno s pomočjo OSP interaktivno izbrati vse potrebne opcije povezovalnika za optimalno sestavo slike programa.



Slika 4: avtomatska priprava komandnih datotek za testiranje programa
Fig 4: automatic prepare of command files for program testing

- segmentna datoteka

Pri povezovanju modulov programa se velikokrat uporablja tudi segmentna datoteka. Pri obsežnejših programih, kjer običajno zmanjka spominskega prostora je potrebno uporabiti segmantno tehniko povezovanja modulov.

Temu namenu služi poseben opisni jezik (overlay discription language), s katerim povezovalniku posredujemo drevesno strukturo našega programa. OSP omogoča opis večine drevesnih struktur v segmentni datoteki.

2.2 Sistem izvornih datotek

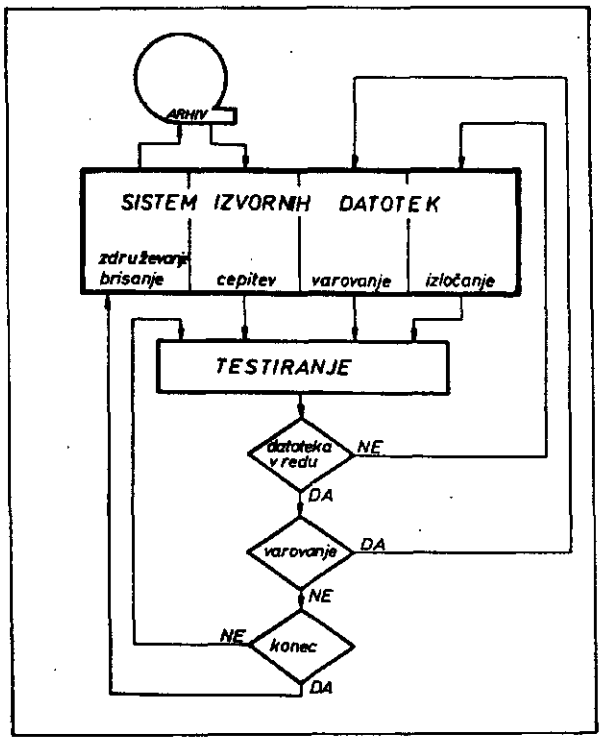
Za rokovanje z izvornimi datotekami programa je bil uveden sistem izvornih datotek. Z njegovo pomočjo dosežemo naslednje cilje:

- avtomatsko rokovanje z izvornimi datotekami
- zaščita vseh datotek programa
- enostavno arhiviranje programa
- enostavno dearhiviranje programa

Vsaka izvorna datoteka ima svojo oznako, ki je shranjena v bazi podatkov. Sistem komandnih datotek omogoča tekoče spremljanje spreminjanja izvornih datotek z vstavljanjem novih oznak, spreminjanjem trenutnih oznak in brisanjem nepotrebnih oznak.

Tako se tekoče zbirajo v bazi podatkov vsi podatki po razsežnosti razvijajočega programa. Programerju je omogočen v vsakem trenutku celovit pregled nad trenutnim stanjem programa. Vse trenutne izvorne datoteke, ki jih je sistem komandnih datotek ažuriral v bazi podatkov, so avtomatsko dodatno zaščitene.

Funkcije sistema izvornih datotek so prirejene posebnostim posamezne faze programiranja (odslej kar OSP).



Slika 5: avtomatsko rokovanje z izvornimi datotekami
Fig 5: automatik source files handling

- faza testiranja programa

Testiranje je najobsežnejša faza programiranja. V tej fazi nastajajo in se neprestano dopolnjujejo datoteke programa. OSP omogoča dva pomembna in tudi najpogostejša posega:

- varovanje: avtomatsko združevanje vseh datotek programa v zaščitno datoteko
- izločanje: izgubljeno ali pokvarjeno datoteko lahko ponovno interaktivno izločimo iz zaščitne datoteke in se tako izognemo dolgotrajnemu vtipkavanju.

- faza arhiviranja programa

Velik problem predstavlja arhiviranje programa. Ob koncu testiranja programa je potrebno shraniti veliko število različnih izvornih datotek v arhiv. Ob večjem številu datotek programa, zlasti še ob večjem številu končanih programov, je to zelo težko rešljiv problem. OSP odpravlja ta problem s posegoma:

- združevanje: ob koncu testiranja je možno vse komandne, izvorne in pomožne datoteke združiti in pri tem vnesti še dodatne podatke za arhiviranje
- brisanje: ko so vse datoteke programa združene v enotni arhivski datoteki, OSP avtomatsko zbriše vse datoteke iz spomina računalnika

Ob koncu testiranja se avtomatsko izgubi sled za pravkar napisanim programom. Slika programa se skopira na uporabniški del spomina, arhivska datoteka programa pa v arhiv.

- faza spreminjanja programa

Vzdrževanje ali prilagajanje že napisanega in testiranega programa uporabniku zahteva neprestano spreminjanje in dopolnjevanje že končanih in arhiviranih programov. Izvedba dearhiviranja izvornih datotek programa temelji na predhodnem načinu arhiviranja. OSP omogoča z nekaj preprostimi posegi ponovno vzpostavitev takega stanja, ki spet omogoča nemoteno testiranje programa.

- cepitev: vse datoteke programa se ponovno izločijo iz arhivske datoteke; program je takoj uporaben za testiranje.

3. ZAKLJUČEK

Programski paket OSP je sestavni del širšega informacijskega sistema za programiranje in služi predvsem zmanjševanju napora pri programiranju in hitrejšemu razvoju tudi bolj kompleksne programske opreme.

Pri večletni uporabi se je dopolnjeval in predstavlja v sedanji obliki nepogrešljiv pripomoček pri razvijanju programov:

- neizkušenemu programerju omogoča hitro vživljanje v zapleten svet programiranja
- izkušenemu programerju omogoča pisanje obsežnih programov, istočasno razvijanje več programov, pisanje programskega paketa
- skupini programerjev omogoča preprostejše sodelovanje pri skupnem programiranju.

Literatura

1. DIGITAL, Reference Manuals RSX-11M v 4.0, Utilities, Task builder, I/O Operations, 1981
2. Kocuvan F., Delovni standardi programiranja, Delta informator 5, 1981, Ljubljana
3. SIS102, Navodila za programiranje, 1984, Ravne

S T A N D A R D I V R A Č U N A L N I S T V U

Neda K A R B A

ŽELEZARNA JESENICE

UDK: 681.3.012

V referatu so opisani standardi programiranja za različne vrste računalnikov. Zaradi standardov programiranja programi niso več odvisni od programerja. To je važno tudi zaradi tega, ker imamo v naši delovni organizaciji tri različne vrste računalniških sistemov in nekateri podatki se uporabljajo na vseh treh sistemih.

STANDARDS IN COMPUTERISATION. In manuscript are described the standards of programming for different type of computers. For the sake of standards of programming the programms are not depended upon the programmers. It is so important because in our firm we have three different type od computer and many times the data are used in all three systems.

UVOD

Danes se že dogaja, da v eni delovni organizaciji uporabljajo različne vrste računalnikov. Na eni strani je to večji komercialni računalnik, nato so lahko še manjši komercialni računalniki (mini ali mikroračunalnik) in procesni računalnik za vodenje različnih procesov. Simultana uporaba takega sklopa različnih vrst računalnikov zahteva vpeljavo določenih standardov v posamezne obdelave. Kajti vhodni podatki enega sistema so lahko izhodni rezultati drugega.

In prav s takim problemom - uporabo različnih vrst računalnikov smo se srečali v Železarni Jesenice kjer sem zaposlena. Imamo komercialni računalnik iz družine IBM 370/3031 (v bližnji prihodnosti dobimo še enega), s 5 MB pomnilnika. Nanj je vezana množica terminalov po vsej železarni. Za spremljanje proizvodnje v eni izmed temeljnih organizacij (v prihodnosti pa še v dveh) uporabljamo miniračunalnik IBM S/1 (512 KB pomnilnika). V bližnji prihodnosti pa bomo instalirali še procesna računalnika PDP 11/44 za vodnje procesa v jeklarni in krmiljenje porabe električne energije. Informacijski sistem Železarne predvideva povezavo vseh treh vrst računalnikov v integralni sistem.

Da bomo lahko vskladili delovanje vseh treh dokaj različnih sistemov smo se že pred časom odločili, da bomo pričeli s pisanjem " Knjige standardov ". V njej naj bi se nahajalo vse, kar zadeva delovanje in programiranje računalnikov v naši delovni organizaciji.
Knjiga ima zaenkrat tri velike (glavne) dele:
- Računalnik IBM 370/3031
- Miniračunalnik IBM S/1
- Procesni računalnik PDP 11/44

Vsak od teh delov pa je dalje razdeljen na posamezna poglavja.

RAČUNALNIK IBM 370/3031

En računalnik IBM 370/3031 uporabljamo v našem računskem centru že od leta 1979, drugega pa bomo predvidoma začeli uporabljati še letos. Zato je prav del, kjer obravnavamo ta tip računalnika v " Knjigi standardov " najbolj obdelan. Posamezna poglavja so dalje razdeljena na podpoglavja. Ta del knjige obravnava tako delovanje kakor tudi uporabo tega računalnika in je razdeljen na naslednja poglavja:
- Organizacija sistema (opis računalniške opreme in konfiguracije sistemov).

- Sistemska programska oprema (opis kupljenih oziroma najetih programskih produktov z navodili za uporabo).
- Lastni programi in podprogrami za splošno uporabo (opis, namen uporabe in navodila za uporabo).
- Navodilo oddelka za obdelavo (organizacija in izvajanje obdelav, planiranje obdelav, terminiranje obdelav, logični potek (zaporedje) obdelav, evidenca trakov, izpisi na tiskalniku (različni formularji), navodila za testiranje obdelav, šifre programerjev in obdelav, dokumentacija področja obdelave po testiranju,...).
- Zaščita in obnavljanje (sistemskih in aplikacijskih bibliotek, uporabniških katalogov, permanentnih datotek, zavarovanje in obnavljanje celotnega področja obdelave - programi,jobi, parametri, prenos iz faze testiranja v produkcijo in nazaj,...).
- Vhodna in izhodna dokumentacija (seznami vhodnih podatkov, seznami izhodnih tabel in njihovih prejemnikov, navodilo za pripravo podatkov za naslovnike tabel, šifrant obrazcev in tipov papirja,...).
- Navodila za batch programiranje (določanje imen datotek, tabel, programov in podprogramov, nosilcev podatkov, jobov, generacij grup, uporaba cond parametra, prevajanje in linkanje programov glede na vrsto programskega jezika (PL/1, cobol, fortran, assembler), uporaba biblioteke za dokumentiranje obdelav (PRD.DOCLIB)).
- Banka podatkov (standardi imen DBD, PSB in segmentov, kreiranje bank, fizične banke podatkov, vzdrževanje, zaščita in obnavljanje banke podatkov, uporabo modulov po pristopih do segmentov, programerski pristopi do bank podatkov, procedure za ažuriranje segmentov bank podatkov).
- VSAM datoteke (definicija in uporaba VSAM datotek, njihova zaščita in obnavljanje).
- Programiranje v ON-LINE (splošen opis, hierarhična odvisnost (področja, aktivnosti, programi, ekrani in akcijske kode), kategorije subjektov in njihove definicije, imena podatkov, programov, uporabniških modulov in ekranov, standardiziranje akcijskih kod, avtomatično generiranje faz, vrste on-line programov, konverzacijsko in nekonverzacijsko programiranje,...).
- Testiranje on-line programov (splošna navodila).
- Produkcija on-line programov (splošna navodila).
- Arhiviranje in restart (batch in on-line obdelav, prehod leta,...).
- Biblioteke (vrste bibliotek in njihove značilnosti).
- Standardi sistemske analize (faze in princip dela, navodila za pripravo priročnika za uporabnika).

MINI RAČUNALNIK IBM S/1

V planu imamo uporabo treh računalnikov tega tipa in sicer v treh različnih temeljnih organizacijah naše železarne. S poskusno proizvodnjo bomo začeli v avgustu na enem izmed teh računalnikov. Z njim bomo spremljali proizvodnjo in sicer zaenkrat predvsem s finančnega stališča. Računalnik je povezan tudi s sistemom IBM 370/3031 tako, da lahko sprejema določene podatke od njega oziroma so nekateri njegovi izhodni podatki vhodni za določene obdelave na tako imenovanem host (glavnem - komercialnem) računalniku.

Ker bo prvi računalnik IBM S/1 šele poskusno obratoval je jasno, da tudi standardi zanj še niso dokončni. Tekom obratovanja se bodo verjetno pokazale določene zahteve, ki jih zaenkrat še nismo predvideli. Za zdaj je ta del knjige standardov razdeljen šele na šest poglavij. Tudi povezave med njim in glavnim računalnikom še niso dokončne. Pomembno je to, da povezave po telefonski liniji med obema računalnikoma lahko tečejo in za zdaj še ni bilo posebnih težav. Tudi tu opisana poglavja obravnavajo delovanje in uporabo miniračunalnika.

- Opis sistema in njegova organizacija (računalniška oprema, konfiguracija sistema, sistemska programska oprema in njena uporaba).
- Programska navodila (določevanje imen programov, podprogramov, jobov, ekranov in datotek, lociranje in velikost datotek, pristopi do datotek, prevajanje in linkanje programov, testiranje obdelav in podobno).
- Standardi sistemske analize (priprava priročnika za programerja in uporabnika, navodila za določevanje imen sistemske analize, imena tabel, plan in izgled ekrana in tabele).
- Organizacija obratovanja računalnika (sistem dela, terminski plan,...).
- Uporabnik in sistem (navodila za uporabnika - pristop do sistema, samo delo s terminali in sistemom. Treba je vedeti, da ta miniračunalnik ne potrebuje posebnega operaterja in da z njim lahko direktno dela uporabnik).

- Zavarovanje in obnavljanje (navodila za zavarovanje podatkov in programov, obnavljanje le-teh, če pride do različnih napak).

Iz vsega navedenega je razvidno, da del " Knjige standardov ", ki pokriva miniračunalnik še ni zaključen in da bo treba še precej dela.

PROCESNI RAČUNALNIK PDP 11/44

V drugi polovici letošnjega leta naj bi pričel z delom tudi procesni računalnik. Zato smo pričeli s pripravo tretjega dela " Knjige standardov ", ki bo obravnaval delo in uporabo procesnega računalnika v Železarni Jesenice. Pravzaprav je treba govoriti o dveh procesnih računalnikih istega tipa, ki bosta vezana paralelno in bosta opravljala pravzaprav vsak svojo funkcijo, istočasno pa bosta tudi varovala drug drugega. To pomeni, da bo v primeru okvare enega procesnega računalnika prevzel drugi njegove najvažnejše naloge. En računalnik bo vodil metalurški proces v jeklarni, drugi pa bo krmilil porabo električnega toka v naši delovni organizaciji. Tudi v tem delu " Knjige standardov " govorimo v nekaj poglavjih in sicer:

- Organizacija sistema (opis računalniške opreme in konfiguracije s pripadajočo sistemsko programsko opremo, namen in način njene uporabe).
- Programska navodila (imena datotek, programov, jobov, organizacija in pristop do datotek, lociranje,...).
- Standardi sistemske analize (priprava priročnika za programerja in uporabnika, imena, plan in izgled ekranov in tabel).
- Organizacija obratovanja procesnega računalnika (način obratovanja, terminski plan,...).
- Uporabnik in sistem (navodila uporabniku za direktno delo s procesnim računalnikom).
- Zavarovanje in obnavljanje (navodila za zavarovanje podatkov in programov obnavljanje le-teh ob raznih napakah).
- Povezava procesnega računalnika z računalnikom IBM 370/3031 preko telefonske linije (navodila za povezavo in način dela).

ZAKLJUČEK

Jasno je, da obstoječa " Knjiga standardov " še ni dokončna in da to verjetno tudi nikoli ne bo. Vedno se bodo z nadaljevanjem dela na do sedaj znanih treh različnih sistemih računalnika pokazale še nove naloge, ki bodo zahtevale nove rešitve tudi na področju standardiziranja tako v programerskem delu kakor tudi v sami uporabi računalnikov. Verjetno pa je pomembno predvsem to, da je tako " Knjiga standardov " potrebna tudi tam, kjer imajo zaenkrat le en računalniški sistem. Omogoča namreč lažje delo vseh njegovih uporabnikov, tako v računskem centru kakor tudi uporabnikov izven centra. To je pravzaprav mesto kjer so zapisane vse tiste značilnosti posameznega računalnika, ki jih potrebujemo pri programiranju in njegovi splošni uporabi. Zato je vpeljava določenih standardov v vsakem računskem centru verjetno dokaj nujna.

POENOSTAVLJENA NAPOVED NASTOPA NAPAK PRI TESTIRANJU PROGRAMSKE OPREME

Tomaž Dogša, dipl. ing., dr. Ivo Rozman
TEHNIŠKA FAKULTETA MARIBOR
JUGOSLAVIJA

UDK: 681.3.06

POVZETEK - V članku je prikazana praktična in enostavna metoda za predikacijo nastopa napak v zadnjem obdobju testiranja. Ocenjen potrebni čas testiranja za odkritje naslednjih napak nam omogoča objektivno odločitev o prenehanju testiranja. Prikazan je tudi konkreten primer predikcije, ki se je uporabil pri testiranju simulatorja teleinformacijskih sistemov v železniškem prometu.

SIMPLE METHOD FOR SOFTWARE FAILURE PREDICTION - A practical and simple method for software failure prediction in the last period of software testing in presented. Objectiv decision when to stop with testing, is possible with results of prediction. A practical example of prediction that was used by software testing of computer railway control systems is also included.

UVOD

Eden izmed glavnih kazalcev kvalitete programske opreme je njena zanesljivost. S testiranjem programov skušamo dobiti odgovore na naslednja vprašanja:

a) Koliko je vseh napak v programu?
b) Kolikšna je zanesljivost delovanja programa v realnem okolju?
c) Kako dolgo je potrebno še testirati, da bo odkritih npr. 90 % vseh napak?
d) Kolikšni bodo stroški testiranja?

Danes so že razviti določeni matematični modeli /2/, /5/, ki bolj ali manj uspešno dajejo odgovore na ta vprašanja. Predvsem v praksi pogosto nerealne predpostavke, na katerih bazirajo ti modeli in pa vpletenost čisto subjektivnega faktorja otežuje deterministični pristop. (Pogoste predpostavke: število vseh napak v programu je konstantno, velikost programa se ne spreminja, večina napak je odpravljenih itd.)

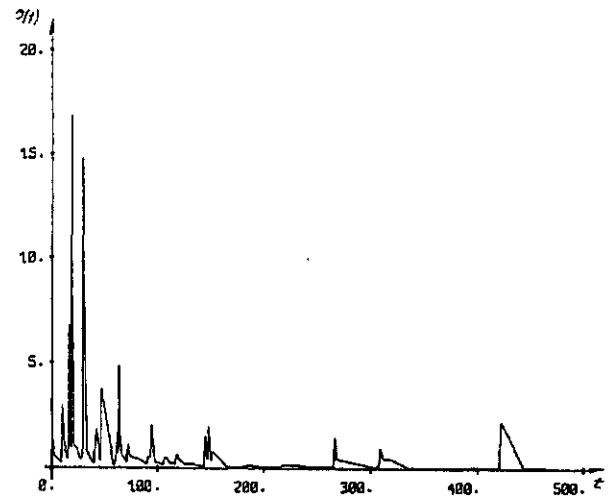
V nadaljevanju članka je prikazano, kako se da brez uporabe zapletenih modelov odgovoriti na vprašanje d). Ta metoda bo ilustrirala še s konkretnim primerom testiranja.

POENOSTAVLJENA PREDIKCIJA

Pri uporabi vsakega modela potrebujemo podatke o rezultatih testiranja. V ta namen je bila razvita posebna procedura /4/, ki je omogočala zbiranje najosnovnejših podatkov:

1. CPU čas potrebnega odkritja napake
2. število odkritih napak (v primeru, da jih je več kot 1)
3. koledarski čas potreben za odkritje napake
4. čas potreben za odpravo napake
5. velikost programa, ki ga testiramo.

Iz teh podatkov se dobijo osnovni pokazatelji, ki jih potrebujemo pri zasledovanju uspešnosti testiranja. V konkretnem primeru smo opazovali intenzivnost pojavljanja napak v odvisnosti od skupnega časa testiranja (slika 1) in pa kumulativno število napak v odvisnosti od CPU časa testiranja (slika 2) /2/.



Slika 1: Intenzivnost pojavljanja napak $\varphi(t)$ v odvisnosti od skupnega CPU časa testiranja t.

Z interpolacijo grafa na sliki 2 dobimo potreben čas za odkritje nekaj naslednjih napak (slika 3). Ta pristop temelji na naslednjih predpostavkah:

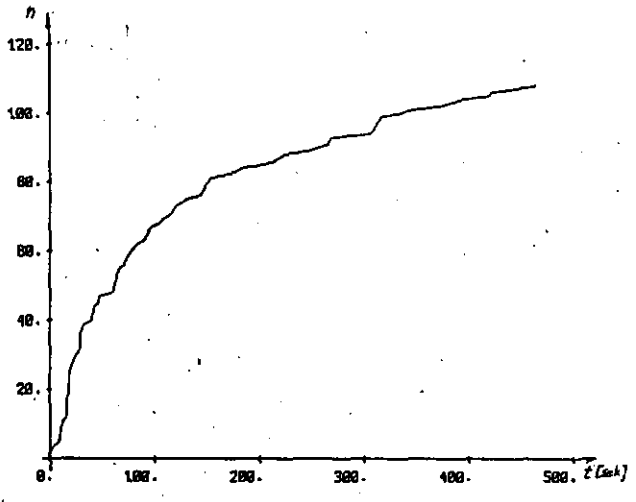
1. večina napak je že odkritih
2. intenzivnost pojavljanja napak ne bo naraščala ampak padala.

Za interpolacijsko funkcijo je bila zbrana

$$f(x) = k_2\sqrt{x} + k_1.$$

Za napoved potrebnega časa Δt, v katerem bomo našli še ΔN napak, moramo določiti enačbo, ki ponazarja

graf na sliki 2. S pomočjo metode povprečnih vrednosti



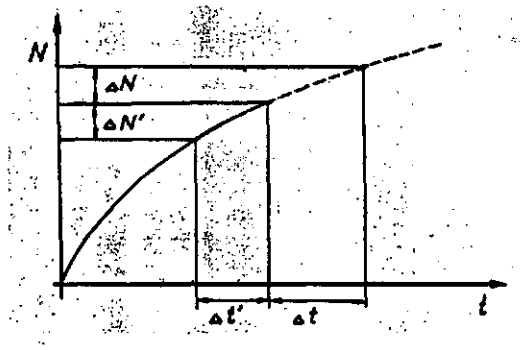
Slika 2: Kumulativno število napak n v odvisnosti od CPU časa testiranja t.

določimo koeficiente interpolacijske funkcije za interval $\Delta N'$. Za $\Delta N'$ naj velja:

$$\Delta N' \geq \Delta N.$$

Ko je funkcija f(x) določena, lahko izračunamo nastop napak v intervalu Δt.

Tabela na sliki 4 prikazuje del numeričnih rezultatov, ki smo jih dobili pri testiranju simulatorja procesov v že-



Slika 3: Preprosta predikcija.

lezniškem prometu. Ko smo odkrili 85 napako, smo napravili predikcijo za nastop naslednjih 5 napak. Napovedan potreben čas testiranja je bil 48,5 sek.(CPU), kasneje izmerjen pa 40,9 sek. Razlika je znašala 18,6 %.

Iz primerjave med napovedanimi podatki in dejanskimi je razvidno, da izbrana interpolacijska funkcija f(x) dobro ustreza. Maksimalno odstopanje na celotnem intervalu je bilo 4 %. Napoved za daljši interval je manj zanesljiva. Če bi v prejšnjem primeru povečali interval ΔN in $\Delta N'$ na 10, bi maksimalno odstopanje naraslo na 10,4 %.

Ker izbrana interpolacijska funkcija nima limitne vrednosti, se analitično ne da določiti število vseh napak. Slednje bi omogočila izbira funkcije

$$f(x) = k_2 (1 - e^{k_1/x}),$$

ki je podlaga znanemu Musovemu modelu /6/. V tem primeru bi postalo določanje koeficientov k_1 in k_2 precej zapleteno, saj bi morali reševati transcendentne enačbe.

t	t^*	N		odstopanje	
150,9	146,49	80		4,41	
152,2	156,25	81		-4,05	
165,7	166,33	82	$\Delta N'$	-0,63	
177,5	176,72	83		0,78	
183,1	187,43	84		-4,33	
201,8	198,45	85		3,35	
213,9	207,79	86		4,11	
219,0	221,44	87		-2,44	
224,3	233,41	88	ΔN	-9,11	(-4 %)
244,7	245,69	89		-0,99	
254,8	258,29	90		-3,49	

$\Delta t = 40{,}9$ $\Delta t' = 48{,}5$ $\Delta N = 5$

Slika 4: Primerjava med predikacijo t^* in dejanskimi rezultati t za napake od 86 do 90. Za kontrolo je izračunano odstopanje interpolacijske krivulje v področju že dobljenih rezultatov $\Delta N'$.

ZAKLJUČEK

Z izbiro preprostega matematičnega modela se da dokaj natančno napovedati nastop napak v zadnjem obdobju testiranja. S temi podatki lahko zelo dobro ocenimo nadaljnje stroške testiranja. Z opazovanjem intenzivnosti pojavljanja napak lahko sklepamo o umirjanju pojavljanja napak. Velika nihanja pomenijo, da s testiranjem vsekakor ne smemo končati.

LITERATURA

/1/ Myers J. G.: The Art of Software Testing, John Wiley and Sons, Inc., New York 1979

/2/ J. Virant: Zanesljivost računalniških sistemov, Zal. Fakulteta za elektrotehniko v Ljubljani, Ljubljana 1981

/3/ M. L. Schooman: Software Reliability: A Historical Perspective, IEEE Trausactions on Raliability, 1984 april, Vol. R-33, No 1, str. 48-55

/4/ T. Dogša: Testiranje programske opreme teleinformacijskih sistemov za vodenje železniškega prometa, Magistrska naloga, Tehniška fakulteta v Mariboru, Maribor 1985

/5/ Alan N. Sukert: Empirical Validation of Three Software Error Prediction Models, IEEE Traus. on Reliability, Vol. R - 28, No. 3, August 1979, str. 199-204

/6/ J. D. Musa: Validity of Execution - Time Theoriy of Software Reliability, IEEE Traus. on Reliability, Vol. R - 28, No. 3, August 1979, str. 181-191

AVTOMATSKO GENERIRANJE TESTNIH ODLOČITVENIH DREVES V SIGNATURNI ANALIZI

ANDREJ DOBRIN, FRANC NOVAK; Institut Josef Stefan
Ljubljana, Jugoslavija

UDK: 681.3.517.11

Ročno generiranje testnih odločitvenih dreves v signaturni analizi zahteva stalno prisotnost visoko kvalificiranega strokovnjaka. Zaradi obsežnosti odločitvenih dreves pogostokrat prihaja do semantičnih napak v strukturi odločitvenega drevesa, ki znatno zmanjšujejo ali celo onemogočajo lokalizacijo virov napak v delovanju preizkušanca. Avtomatsko generiranje odločitvenih dreves odpravlja semantične napake, poleg tega pa nudi precej ugodnosti s stališča prilagajanja testiranega vezja za testabilnost. Opozarja tudi na morebitne nedorečenosti uporabljenega vzbujanja preizkušanca.

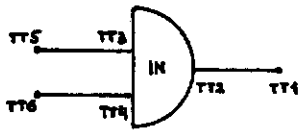
AVTOMATIC GENERATION OF DECISION TREES IN SIGNATURE ANALISYS - Manual generation of decision trees in signature analisys demands for hish qualified personell. Due to the complexity of the decision trees it is impossible to avoid human errors, which affect the reliability of the used troubleshooting method for localization of the detected errors. The use of the program peckage that allows the avtomatic generation of the decision trees overcomes those troubles. Besides it also give s the possibility to increase the design for testability in the tested product by pointing untestable areas of tested circuits.

Lokalizacija virov napak v delovanju preizkušancev z metodo signaturne analize poteka s primerjanjem signatur v testnih točkah. Referenčne signature se določajo na pravilno delujočem modulu. Vsi pogoji, kot na primer način vzbujanja testiranega vezja, priključitev sond signaturnega analizatorja, izbira testnih pogojev, itd., ki so veljali med odjemanjem referenčnih signatur z "zlate plošče", morajo biti izpolnjeni tudi med potekom testiranja tekočega preizkušanca.
Neenakost referenčne in odčitane signature v isti testni točki opozarja na napako, ki se nahaja nekje med opazovano testno točko in funkcijskimi viri signalov, ki definirajo podatkovni pretok v testni točki.
Detektiranemu napačnemu podatkovnemu toku sledimo preko testnih točk, ki se nahajajo na električnih poteh signalov, ki tvorijo dani podatkovni pretok. V trenutku, ko odčitane signature postanejo enake referenčnim, smo prešli preko vira detektirane napake. Napačno delujoč element se nahaja na tako določeni meji med dvema zaporedno odčitanima testnima točkama.
Testne točke, ki predstavljajo skrajne ponore podatkovnih pretokov definirano vzbujanega preizkušanca, imenujmo ključne točke vezja. Preizkušanec pravilno deluje pri danem vzbujanju, kadar signature v vseh ključnih točkah vezja ustrezajo referenčnim signaturam.
Detektirani napaki v ključni točki sledimo preko testnih točk, ki se nahajajo na podatkovnih poteh v obratni smeri podatkovnih pretokov. Izbira naslednje testne točke je odvisna od strukture testiranega vezja in rezultata primerjanja signatur. Naslednje testne točke izbiramo vse dotlej, dokler v odvisnosti od skladanja signatur z referenčnimi ne definiramo področja med dvema testnima točkama, kjer se nahaja vir napake.
Povsem jasno je, da je nepotrebno odčitavanje signatur v testnih točkah, ki se nahajajo na podatkovnih poteh, kjer ima skrajna testna točka ustrezno signaturo.
Opis lokalizacijskega postopka najlaže prikažemo v obliki binarnega drevesa, ki ga imenujemo odločitveno drevo. Korenino odlo-

čitvenega drevesa predstavlja ključna točka. V kolikor celotnega delovanja preizkušanca ni mogoče zaznati v eni sami testni točki, ustreza vsaki posamezni ključni točki pripadajoče poddrevo. Zaporedje odčitavanja je poljubno in je izbrano največkrat zgolj s stališča najenostavnejšega odčitavanja signatur v ključnih testnih točkah. Vozlišče odločitvenega drevesa je podano s testno točko in njeno pripadajočo signaturo. Vejitev na naslednika je pogojena z rezultatom primerjanja detektirane signature z referenčno vrednostjo. Listi odločitvenega drevesa predstavljajo diagnoze napak, ki so definirane z opravljeno potjo preko testnih točk, kjer so bile detektirane neustrezne signature.

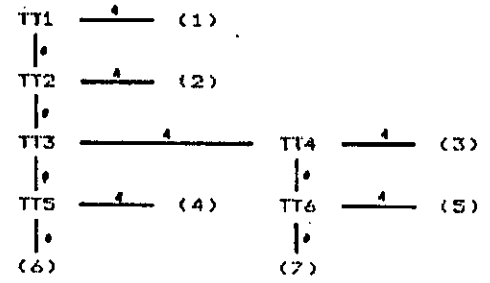
Grajenje testnega odločitvenega drevesa se prične z določitvijo ključnih testnih točk preizkušanega vezja pri definiranem vzbujanju in enolično določenih in izpolnjenih pogojih delovanja, kot na primer nastavitev stikal, prevoz in podobno. Za definicijo ključnih točk je potrebno podrobno poznavanje delovanja vezja,kakor tudi delovanje posameznih elementov vezja in njihovih medsebojnih povezav. Ključnim točkam sledé opisi nadaljnih testnih točk z definicijami vejitev v primeru ustrezne oziroma neustrezne signature. Vsaka izbrana testna točka mora biti neposredno dostopna na površini tiskanine preizkušanca. Potrebno je paziti na ustrezno zaporedje predpisanih testnih točk. Kadar posamezna vejitev iz testne točke nima nadaljevanja v nobeni drugi testni točki je potrebno določiti diagnozo napake. Diagnoza napake je odvisna od pogojev, ki so povzročili ustrezne vejitve v odločitvenem drevesu. Grajenje odločitvenega drevesa je končano, kadar so vsi možni sprehodi po drevesu zaključeni z ustrezno diagnozo. Kadar pride med grajenjem testnega drevesa do konfliktnih situacij, kot na primer zelo obširna in zategadelj relativno neuporabna diagnoza, vejitev na že opisano testno točko, ki se nahaja na višjem nivoju kot opisovana testna točka (detektirana napaka v tem primeru propagira v zanki) in podobno, je potreben povraten poseg v definicijo testnih pogojev. V najenostavnejšem primeru pomeni takšen poseg ponovno ustreznejšo definicijo predpisanega vzbujanja. Zadovoljivo rešitev dostikrat predstavlja tudi ponovna definicija krmilnih signalov, s katero zadovoljimo zahteve testne metode signaturne analize. Večje težave nastopijo v primeru, kadar naletimo na povratne povezave v testiranem vezju. Edino možno pot predstavlja v tem primeru poseg v samo shemo testiranega vezja, saj je potrebno ustvariti razklenitev povratne podatkovne zanke in na ta način ustvariti enosmeren podatkovni pretok. Opisane neugodnosti se pojavljajo še zlasti med izdelovanjem testnih postopkov preikušencev, kjer v fazi načrtovanja vezja niso bila upoštevana vsaj najsplošnejša pravila o testibilnosti vezij.

Iz navedenega je razvidno, de je za definicijo odločitvenega testnega drevesa potrebno obširno znanje, ki ga ima le visoko usposobljen kader. Prav tako je tudi razvidno, da sama izdelava testnega drevesa zahteva obilo zamudnega in nekreativnega ročnega dela, kot na primer vpisovanje posameznih testnih točk z ustreznimi signaturami. Pri tem se moramo zavedati, da ob delu nujno prihaja do napak v sintaksi odločitvenega drevesa, katerih edini razlog je človeški faktor. Napaka v strukturi ali posameznih zapisih testnega drevesa nujno povzroči neustrezno diagnozo detektirane napake in s tem neuporabnost testnega postopka za lokalizacijo vira napake. Grobe napake v zgradbi odločitvenega drevesa je ob pazljivem pregledovanju njegove strukture možno odpraviti še pred uporabo odločitvenega drevesa za testiranje preizkušancem, medtem, ko manj opazne napake ostanejo. Zaradi obsežnosti večine odločitvenih dreves skoraj ni mogoče preizkusiti njihovo pravilnost s simuliranjem napak na preizkušanem vezju. Na dan prihajajo med samim postopkom testiranja. Odpravljamo jih lahko le postopoma, pri čemer pa se je potrebno zavedati, da pri operaterju mnogo prej pride do nezaupanja v sam testni postopek, kot pa do eliminacije vseh obstoječih napak v odločitvenem drevesu. Slika 1 prikazuje običajna IN vrata in pripadajoče odločitveno testno drevo za lokalizacijo vira napake.



Ključna testna točka : TT1
Podatkovni izvor : TT5, TT6

Legenda : 0 - nepravilna signatura
1 - pravilna signatura



Diagnoze :

(1) : Pravilno delovanje testiranega vezja
(2) : Napaka na povezavi TT2 - TT1
(3) : Napaka v delovanju IN vrat
(4) : Napaka na povezavi TT5 - TT3
(5) : Napaka na povezavi TT6 - TT4
(6) : Napaka v vhodnih podatkih
(7) : Napaka v vhodnih podatkih

SLIKA 1

Prvenstveni namen avtomatskega generiranja odločitvenih dreves v signaturni analizi je eliminacija sintaktičnih in logičnih napak v strukturi odločitvenih dreves. Na ta način odločilno vplivamo na faktor zapanja v testni metodi. Ustrezna dodatna programska oprema poskrbi tudi za drastično zmanjšanje potrebnega časa za izdelavo testnih postopkov. Celotno programsko opremo lahko razdelimo na več podsklopov in sicer:

- generiranje datoteke testnih točk, ki opisuje elemete preizkušanca
- določanje signatur posameznim testnim točkam in generiranje datoteke povezav med testnimi točkami
- preverjanje testabilnosti designa preizkušanca in določevanje ključnih testnih točk
- generiranje testnega - odločitvenega dreves.

Z uporabo postopka avtomatskega generiranja odločitvenih dreves je bistveno zmanjšan potrebni čas sodelovanja visokokvalificiranega kadra, saj precejšen del postopka lahko opravlja nižje kvalificirani kader. Večino potrebnega strokovnega dela lahko opravi načrtovalec vezja sam, hkrati pa ob upoštevanju diagnoz programskega paketa doseže, da je izdelano vezje prilagojeno zahtevam načrtovanja za testabilnost. S takšnimi povratnimi posegi je mogoče doseči na celotnem vezju lokalizacijo vira napake do elementa ali povezave natančno, kar pohitri tudi postopek oživljanja nepravilno delujočih preizkušancev v kasnejši proizvodnji.

Generiranje datoteke testnih točk, ki opisuje testirano vezje predstavlja opis posameznih elementov vezja. Opis posameznega elementa je poenostavljen do te mere, da je za nožico elementa, ki predstavlja testno točko, potrebno podati le njeno funkcijo v smislu izvora ali ponora podatkov. Opisi večine najbolj splošnih integriranih vezij so podani že v testni knjižnici, tako da je za opis takšnega elementa potrebno navesti le njegovo oznako. Kadar je del elementa neizkoriščen, je možno redundantne testne točke, ki se nahajajo na elementu, enostavno izpustiti. Na ta način krčimo obsežnost datoteke testnih točk, kar zmanjša časovne in prostorske zahteve kasnejše obdelave vpisanih podatkov. Pri kreiranju datoteke testnih točk je pomembno, da med seboj ločimo dele testiranega vezja, ki delujejo med seboj asinhrono, saj asinhronost delovanja onemogoča uspešno uporabo signaturne analize.

Določanje signatur posameznih testnih točk poteka ob natanko definiranem vzbujanju, kjer morajo biti izpolnjeni vsi morebitni dodatni testni pogoji, ki se nanašajo na morebitne posege v testirano vezje in namestitev odjemnih sond signaturnega analizatorja. Operater mora v tem primeru interaktivno vnašati podatke o odčitanih signaturah. Vsaki testni točki, ki se nahaja v datoteki testnih točk, pripišemo ustrezno signaturo. Operater s pomočjo signaturnega analizatorja odčituje signature testnih točk, ki se dodajajo opisom posameznih testnih točk. Ker velja pravilo, da se signature testnih točk, ki se nahajajo na isti električni povezavi, med seboj ne razlikujejo, program sproti preverja enakost odčitane z že prej določenimi signaturami. V primeru, ko naleti v datoteki testnih točk, na testno točko, kateri je pripisana enaka signatura, zahteva od operaterja informacijo o povezanosti oziroma izoliranosti tekoče testne točke z namembno testno točko. V primeru potrditve povezanosti med testnima točkama, se v datoteko povezav vpiše podatek o odkriti povezavi.

Kadar več testnih točk že pripada določeni električni povezavi, je dovolj zgolj ena potrditev povezanosti tekoče testne točke na skupno povezavo. Ko so določene signature vsem testnim točkam v datoteki testnih točk, se v datoteki povezav nahaja popoln opis medsebojne povezanosti ali izoliranosti vseh navedenih testnih točk.
Z definiranjem datoteke testnih točk s pripadajočimi signaturami in datoteke povezav preizkušanca so podani vsi potrebni pogoji, ki omogočajo generiranje odločitvenih testnih dreves. Programskemu paketu je dodan vmesni korak, ki poenostavlja določanje ključnih testnih točk, oziroma opozarja na morebitno netestabilnost posameznih funkcijskih delov testiranega vezja.Ključne točke se določajo na ta način, da izhajamo iz vseh začetnih podatkovnih izvorov in se preko posameznih povezav in elementov vezja pomikamo proti skrajnim podatkovnim ponorom. Testne točke, ki ustrezajo skrajnim podatkovnim ponorom, predstavljajo ključne točke testiranega vezja ob danih testnih pogojih. Kadar je postopek določanja ključnih točk prekinjen zaradi prisotnosti morebitne povratne podatkovne povezave, programski paket javi obvestilo o detektirani podatkovni zanki in zahteva rešitev konfliktne situacije. Le to je mogoče doseči na dva načina. Manj priporočljivo je prekinjanje povratnih povezav na ta način, da odstranimo del vase zaključenega vezja iz datoteke testnih točk, saj s tem oslabimo lokalizacijske sposobnosti testnega postopka. Priporočljivo je, da v takšnih primerih raje spremenimo design testiranega vezja z uvajanjem dodatnih prevez, ki jih lahko med samim postopkom testiranja zaključimo in s tem prekinemo povratne zanke v vezju. Vse morebitne spremembe na preizkušancu je seveda potrebno vnesti tudi v datoteki testnih točk in povezav. Pohitritev postopka definiranja ključnih testnih točk je mogoče doseči z eksplicitno navedbo testnih točk, ki v danem primeru predstavljajo podatkovne izvore. Na ta način lahko hkrati tudi dosežemo eliminacijo asinhrono delujočih delov vezja.
Generiranje odločitvenih testnih dreves predstavlja poslednji korak definiranja testnega postopka določenega preizkušanca.Vsaki ključni testni točki pripada ustrezno odločitveno poddrevo, ki definira morebitni vir napake detektirane v pripadajoči ključni točki. Diagnoze, ki jih postavlja programski paket, popolnoma definirajo tipe napak do katerih pride pri popolni definiranosti vira napak. V primerih, ko vir napake predstavlja nedefinirano velik podsklop testiranega vezja, se postavi splošna diagnoza, ki pa je ošteviločena, tako da lahko operater kasneje popravi tekst posamezne diagnoze.
Največja prednost avtomatsko zgrajenih odločitvenih testnih dreves je v njihovi semantični pravilnosti. Na ta način so odstranjeni vsi možni človeški vplivi, ki zmanjšujejo zanesljivost postopka testiranja. Edina slabost takšnih odločitvenih dreves so diagnoze napak v nezadostno definiranih področjih testiranega vezja. Vendar so tudi te diagnoze posledica nestabilnosti preizkušanca. V vseh primerih, ko je design preizkušanca popolnoma testibilen, so tudi podane diagnoze virov napak enolične in nedvoumne. Že v začetnih korakih programski paket sam opozarja na morebitne nedoslednosti s stališča testabilnosti preizkušanca in odstranitev vzrokov za opozorilo oziroma redesign kritičnih delov vezja v mnogočem popravi lokalizacijske zmožnosti testnega postopka. Dovolj zgodna vključitev programskega paketa v postopek načrtovanja vezij v veliki meri olajša vpeljavo produkta v proizvodnjo.
Programski paket lahko na računalniku z 64K bytov delovnega spomina obdeluje vezja, ki so sestavljena iz približno sto različnih elementov, oziroma 2000 testnih točk. Ker je zelo malo verjetno, da bi z enim samim vzbujevalnim postopkom vzbujali vse elemente preizkušanca, to pomeni, da programski paket obvladuje tudi vezja, v katera je vgrajeno precej večje število elementov.

ITERATIVNA METODA ZA BRZU REALIZACIJU MATEMATIČKIH FUNKCIJA NA ELEKTRONIČKOM RAČUNALU

Milorad Tomić,
DI "Česma" Bjelovar
Jugoslavija

UDK: 681.3.514

U radu je prezentirana iterativna metoda za brzo izračunavanje vrijednosti elementarnih matematičkih funkcija, temeljena na operacijama zbrajanja i pomaka decimalnog zareza. Metoda omogućuje realizaciju tih funkcija u vremenu potrebnom za izvršenje samo jedne do dvije operacije dijeljenja. Izvršena je modifikacija metode u smislu daljnjeg skraćivanja iterativnog procesa za male argumente trigonometrijskih funkcija.

ITERATIVE METHOD FOR QUICK REALIZATION OF MATHEMATICAL FUNCTIONS ON COMPUTER. The paper gives an iterative method for quick calculations of elementary mathematical functions values, based on the operations of addition and decimal point shift. This method makes possible the realization of these functions in the period of time necessary for the accomplishment of only one or two division operations. The author has performed a modification of the method in the sense of shortening still further the iterative process for small arguments of trigonometric functions.

OSNOVNE POSTAVKE ITERATIVNE METODE "CIFRA ZA CIFROM"

Iterativna metoda "cifra za cifrom" zasniva se na postupku tranformacije pravokutnih koordinata vektora u polarne. U geometrijskom smislu metoda koristi "oscilacije" vrha radijus-vektora od zadanog početnog položaja, sve dok ne dostigne traženu vrijednost. U principu, metoda omogućuje istovremenu realizaciju para funkcija

Postavke ove metode bit će prezentirane za slučaj trigonometrijskih funkcija, iako je moguća primjena na sve elementarne (i ne samo njih) matematičke funkcije.

Na primjer, za izračunavanje vrijednosti para funkcija

$$(\sin \varphi , \cos \varphi),$$

odabiremo početni položaj vektora $R_o(1,0)$, rotiramo ga za kutove α_i, $i = 0,1,\ldots$ tako da je algebarska suma tih kutova jednaka argumentu φ.

Nakon izvjesnog (u aproksimaciji konačnog) broja koraka, dobiva se vektor $R(x,y)$. Vrijedi slijedeći odnos:

$$y = \sin \varphi$$
$$x = \cos \varphi .$$

Pošto formule za izračunavanje novih koordinata iz starih glase:

$$y_{i+1} = \cos\alpha_i \, (y_i + x_i \operatorname{tg} \alpha_i)$$
$$x_{i+1} = \cos\alpha_i \, (x_i - y_i \operatorname{tg} \alpha_i),$$

kutove α_i odabiremo tako da je

$$\alpha_i = \operatorname{arc\,tg}\, 2^{-i}$$

i na taj način operaciju množenja s $\operatorname{tg} \alpha_i$ zamjenjujemo množenjem brojem 2^{-i}, što u binarnom brojevnom sistemu predstavlja operaciju pomaka decimalnog zareza.

Izostavimo li u jednadžbama faktor $\cos \alpha_i$, dolazi do povećanja modula vektora. To povećanje kompenziramo odabiranjem pogodne početne vrijednosti apsciso x_o.

Kutove α_i treba pribrajati ili oduzimati, već prema tome da li je njihova prethodna suma veća ili manja od φ. Predznak tih kutova određujemo iz jednadžbe

$$\operatorname{sign} \xi_i = \operatorname{sign} \Big(\varphi - \sum_{j=0}^{i-1} \xi_j \alpha_j \Big).$$

Uvažavanjem prezentiranih postavki, dolazimo do formula algoritma za realizaciju pomenutih funkcija:

$$y_{i+1} = y_i + \xi_i \, x_i \cdot 2^{-i}$$
$$x_{i+1} = x_i - \xi_i \, y_i \cdot 2^{-i}$$
$$\varphi_{i+1} = \varphi_i - \xi_i \operatorname{arc\,tg}\, 2^{-i},$$

uz početne uvjete

$$y_o = 0, \quad \varphi_o = \varphi ,$$

Kadar več testnih točk že pripada določeni električni povezavi, je dovolj zgolj ena potrditev povezanosti tekoče testne točke na skupno povezavo. Ko so določene signature vsem testnim točkam v datoteki testnih točk, se v datoteki povezav nahaja popoln opis medsebojne povezanosti ali izoliranosti vseh navedenih testnih točk.
Z definiranjem datoteke testnih točk s pripadajočimi signaturami in datoteke povezav preizkušanca so podani vsi potrebni pogoji, ki omogočajo generiranje odločitvenih testnih dreves. Programskemu paketu je dodan vmesni korak, ki poenostavlja določanje ključnih testnih točk, oziroma opozarja na morebitno netestabilnost posameznih funkcijskih delov testiranega vezja.Ključne točke se določajo na ta način, da izhajamo iz vseh začetnih podatkovnih izvorov in se preko posameznih povezav in elementov vezja pomikamo proti skrajnim podatkovnim ponorom. Testne točke, ki ustrezajo skrajnim podatkovnim ponorom, predstavljajo ključne točke testiranega vezja ob danih testnih pogojih. Kadar je postopek določanja ključnih točk prekinjen zaradi prisotnosti morebitne povratne podatkovne povezave, programski paket javi obvestilo o detektirani podatkovni zanki in zahteva rešitev konfliktne situacije. Le to je mogoče doseči na dva načina. Manj priporočljivo je prekinjanje povratnih povezav na ta način, da odstranimo del vase zaključenega vezja iz datoteke testnih točk, saj s tem oslabimo lokalizacijske sposobnosti testnega postopka. Priporočljivo je, da v takšnih primerih raje spremenimo design testiranega vezja z uvajanjem dodatnih prevez, ki jih lahko med samim postopkom testiranja zaključimo in s tem prekinemo povratne zanke v vezju. Vse morebitne spremembe na preizkušancu je seveda potrebno vnesti tudi v datoteki testnih točk in povezav. Pohitritev postopka definiranja ključnih testnih točk je mogoče doseči z eksplicitno navedbo testnih točk, ki v danem primeru predstavljajo podatkovne izvore. Na ta način lahko hkrati tudi dosežemo eliminacijo asinhrono delujočih delov vezja.
Generiranje odločitvenih testnih dreves predstavlja poslednji korak definiranja testnega postopka določenega preizkušanca.Vsaki ključni testni točki pripada ustrezno odločitveno poddrevo, ki definira morebitni vir napake detektirane v pripadajoči ključni točki. Diagnoze, ki jih postavlja programski paket, popolnoma definirajo tipe napak do katerih pride pri popolni definiranosti vira napak. V primerih, ko vir napake predstavlja nedefinirano velik podsklop testiranega vezja, se postavi splošna diagnoza, ki pa je ošteviločena, tako da lahko operater kasneje popravi tekst posamezne diagnoze.
Največja prednost avtomatsko zgrajenih odločitvenih testnih dreves je v njihovi semantični pravilnosti. Na ta način so odstranjeni vsi možni človeški vplivi, ki zmanjšujejo zanesljivost postopka testiranja. Edina slabost takšnih odločitvenih dreves so diagnoze napak v nezadostno definiranih področjih testiranega vezja. Vendar so tudi te diagnoze posledica nestabilnosti preizkušanca. V vseh primerih, ko je design preizkušanca popolnoma testibilen, so tudi podane diagnoze virov napak enolične in nedvoumne. Že v začetnih korakih programski paket sam opozarja na morebitne nedoslednosti s stališča testabilnosti preizkušanca in odstranitev vzrokov za opozorilo oziroma redesign kritičnih delov vezja v mnogočem popravi lokalizacijske zmožnosti testnega postopka. Dovolj zgodna vključitev programskega paketa v postopek načrtovanja vezij v veliki meri olajša vpeljavo produkta v proizvodnjo.
Programski paket lahko na računalniku z 64K bytov delovnega spomina obdeluje vezja, ki so sestavljena iz približno sto različnih elementov, oziroma 2000 testnih točk. Ker je zelo malo verjetno, da bi z enim samim vzbujevalnim postopkom vzbujali vse elemente preizkušanca, to pomeni, da programski paket obvladuje tudi vezja, v katera je vgrajeno precej večje število elementov.

$$x_0 = \frac{1}{\sqrt{\prod_{i=0}^{n-1}(1 + 2^{-2i})}}.$$

Nakon n koraka iteracije, dobiva se rezultat

$$x_n = \cos \varphi$$
$$y_n = \sin \varphi ,$$

s točnošću do 2^{-n+1}.

Očito je da u gornjim jednadžbama imamo samo "brze" operacije zbrajanja i pomaka decimalnog zareza.

Da bismo realizirali iterativni postupak, nužno je uz algoritam memorirati konstante

$$\text{arc tg}\, \alpha_i,\ i = 0,1,\ldots,n-1,$$

što je i jedini nedostatak metode. Preimućstva su:

- vrlo brza realizacija algoritama
- jednostavna organizacija numeričkog postupka za realizaciju para funkcija
- efektna ocjena i laka kompenzacija greške
- omogućavanje definiranja relativno malog broja baznih funkcija itd.

MODIFIKACIJA METODE ZA MALE ARGUMENTE

Ukoliko se vrši realizacija para napr. trigonometrijskih funkcija za mali argument φ, potrebno je upotrijebiti izvjestan broj istih iterativnih koraka, u smislu upotrebe kutova α_i u istom redosljedu s istim predznacima.

Vrijedi slijedeći

TEOREM. Neka su brojevi φ i ψ dani u obliku

$$\varphi = m_\varphi\, 2^{-p}, \qquad \psi = m_\psi\, 2^{-p},$$
$$m_\varphi, m_\psi \in [1,2),\ p = 0,1,\ldots,n$$

i neka je iterativni proces definiran kao u prethodnoj točki. Tada se prvih p iterativnih varijabli tih brojeva podudaraju, tj.

$$x_j(\varphi) = x_j(\psi)$$
$$y_j(\varphi) = y_j(\psi),$$

za svako j, $j \le p$.

Da bismo kompenzirali deformaciju trigonometrijskog vektora c_T, uvodimo modificirane početne uvjete

$$x_0(p),\ y_0(p),$$

čije se vrijednosti lako nalaze pomoću standardnog algoritma i koriste se kao konstante uz modificirani algoritam.

Uvodimo nadalje oznaku

$$A(p) = \sum_{i=0}^{p-1} \xi_i \alpha_i .$$

Na taj se način standardni algoritam transformira u modificirani, uz početne uvjete (tj. varijable ulaska u iterativni proces):

$$x_p = x_0(p)$$
$$y_p = y_0(p)$$
$$\varphi_p = \varphi - A(p).$$

Nakon samo n' = n - p koraka iteracije, dolazi se do realizacije razmatranog para trigonometrijskih funkcija.

Treba istaći činjenicu koja proizlazi iz ovako koncipiranog modificiranog algoritma. Naime, pošto se standardni algoritam primjenjuje za vrijednosti argumenta

$$\varphi \in (0, \pi/2),$$

uzimajući u obzir opće postavke modificiranog algoritma, njegova primjena ima smisla za sve argumente

$$\varphi \in (0,1).$$

Napomenimo na kraju da bi implementacijom sistema algoritama za realizaciju elementarnih matematičkih funkcija u vidu standardnih potprograma na elektroničkom računalu, temeljenih na iterativnoj metodi "cifra za cifrom", a posebno na njenoj modificiranoj varijanti, dobili svrsishodniju, napose bržu, realizaciju matematičkih funkcija i njihovih aplikacija u širem smislu.

LITERATURA

1. Avizienis A.,Tung Ch.: A universal arithmetic building element and design methods for arithmetic processors, "IEEE Trans. Comput.", 1970.
2. Bajkov V.D.,Smolov V.B.: Apparaturnaja realizacija elementarnjih funkcij v CVM, Leningrad 1975.
3. Mangulis V.: Handbook of Series for Scientists and Engineers, New York 1976.
4. Rabinovič Z.L.,Ramanauskas V.A.: Tipovye operacii v vyčisliteljnjih mašinah, "Tehnika" Kiev 1980.
5. Tomić M.: Iterativna metoda "cifra za cifrom", Sistemi delta, Ljubljana 1984.
6. Volder J.E.: The Cordic trigonometric computing technique, "IRE Trans. Electronic Comput.", 1979.

PRIMENA MIKRORAČUNARSKOG SIMULATORA TRANSPORTNOG KAŠNJENJA U SMITOVOJ METODI AUTOMATSKE REGULACIJE

Dejan Stajić
Viša škola za primenjenu informatiku i statistiku, Beograd

UDK: 681.3.007

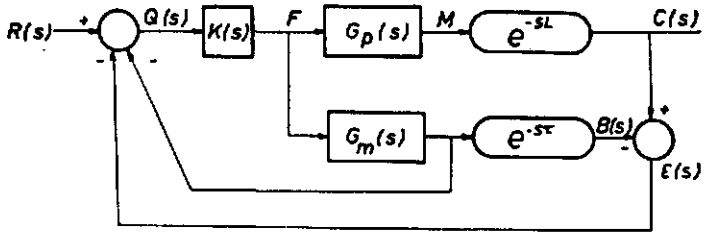
U ovom radu je opisano kako se simulator transportnog (čistog) kašnjenja na bazi mikroprocesora koristi u implementaciji Smitove metode automatske regulacije procesa (dinamičkih sistema) sa dugim transportnim kašnjenjem. Rezultati računarske simulacije sistema automatskog upravljanja jednog hemijsko-tehnološkog procesa pokazuje opravdanost ove primene.

APPLICATION OF MICROPROCESSOR-BASED SIMULATOR OF TIME DELAY TO SMITH'S METHOD OF AUTOMATIC CONTROL. This article describes the use of microprocessor-based simulator in an implementation of Smith's method for automatic control of processes (dynamic systems) with long pure time delays. This application is justified by the results obtained by the computer simulation of the control system for a chemical process.

U V O D

U teoriji automatskog upravljanja poznato je transportno ili čisto vremensko kašnjenje L koje se u domenu kompleksne frekvencije S izražava negativnim izložiocem eksponencijalnog člana (e^{-Ls}). Ono u sistemu automatskog upravljanja izaziva dodatne probleme - usporen odziv i tendenciju ka oscilovanju, tj. nestabilnost sistema /1/. Kod sporih dinamičkih sistema - što je slučaj kod niza procesa u hemijskom inženjerstvu - ovo transportno kašnjenje se sa manje ili više uspeha zanemaruje (korišćenjem konvencionalnog regulatora) ili se na neki način eliminiše njegov uticaj na primer u tzv. predregulaciji ('feedforward'). Medjutim, predregulacija se ne može svuda primeniti, jer ona zahteva tačno izmerene poremećaje i poznavanje tačnog matematičkog modela procesa što nije čest slučaj. Zbog toga je u automatskoj regulaciji procesa sa relativno dugim transportnim kašnjenjem (koje je duže od najduže kapacitivne vremenske konstante procesa) predloženo više načina kompenzacije transportnog kašnjenja. Medjutim, glavni nedostaci ovih načina regulacije su nemogućnost praktične realizacije i neuniverzalnost u primeni. Na primer kod jednog od najboljih načina regulacije - tzv. Smitove metode - najveći je problem praktična realizacija kompenzatora, tj. elementa dugog transportnog kašnjenja.

Imajući u vidu dobre osobine analogne računarske simulacije sistema automatskog upravljanja (projektant ima utisak da radi sa realnim procesom, lako unosi promene parametara procesa i poremećaja i posmatra njihov uticaj na izlaznu veličinu, moguća je električna veza sa realnim regulatorom, mernim pretvaračem, aktuatorom i dr.) u ovom radu je vršena analogna računarska simulacija kako samog proce-



Sl. 1. Blok-šema Smitove metode regulacije

sa tako i ostalih elemenata sistema automatskog upravljanja. Izuzetak čine samo elementi transportnog kašnjenja koji su realizovani pomoću mikroprocesorskih simulatora.

SMITOVA METODA AUTOMATSKE REGULACIJE

Za automatsku regulaciju sistema sa relativno dugim transportnim kašnjenjem L (koje je veće od dominantne kapacitivne vremenske konstante) teoretski je razvijen Smitov princip i metoda /1/ koja omogućava, zahvaljujući specijalnom regulatoru, da je izlazni signal istog oblika kao za sistem bez kašnjenja s tim što vremenski kasni za L.

Ako je prenosna funkcija procesa koji treba regulisati $G_p(s)e^{-sL}$ (s - je kompleksna kružna frekvencija a L - transportno kašnjenje, K(s) - prenosna funkcija konvencionalnog regulatora) tada se prenosna funkcija Smitovog regulatora može predstaviti sledećim izrazom:

$$K^{*}(s) = \frac{K(s)}{1 + K(s)\cdot G(s)\cdot(1 - e^{-sL})}$$

ili u obliku blok-šeme pogodne za praktičnu implementaciju (sl. 1) u kojoj je $G_m(s)e^{-s\tau}$ prenosne funkcije tzv. kompenzatora.

Za ovu šemu se lako izvodi da je izlazni signal procesa

$$C = \frac{KRG_p e^{-sL}}{1 + KG_m + K(G_p e^{-sL} - G_m e^{-s\tau})}$$

pri čemu su sve veličine date u domenu kompleksne frekvencije.

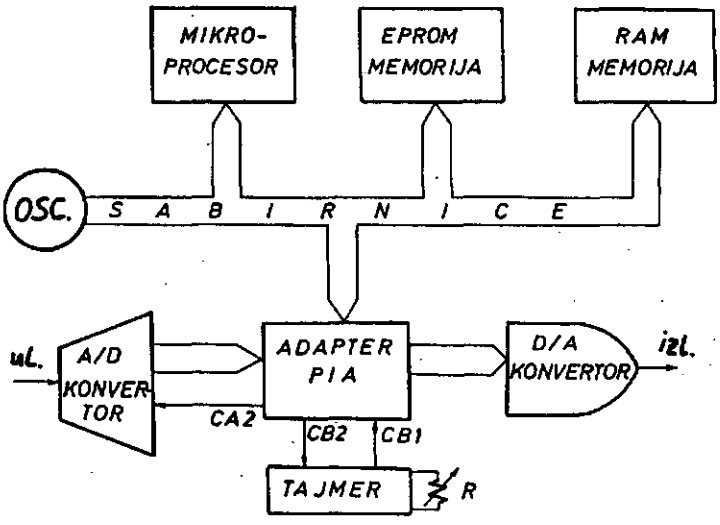
Ako su $G_m = G_p$ i $\tau = L$ tada je funkcija spregnutog prenosa

$$W_s(s) = \frac{K\cdot G_p\cdot e^{-sL}}{1 + K\cdot Gp}$$

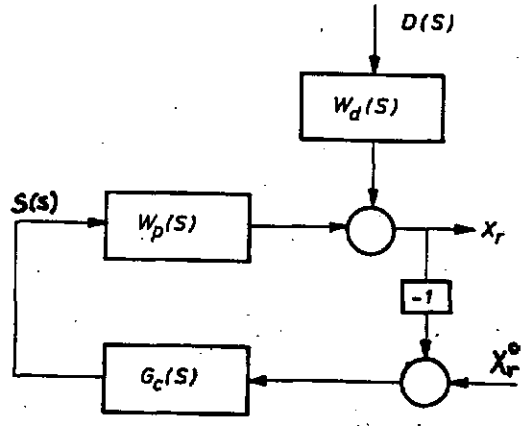
čime je zadovoljen Smitov princip - W_s nema transcendentnih polova pa se može usvojiti dovoljno veliko pojačanje regulatora kako bi se što brže uspostavila stacionarna vrednost izlaznog signala.

Najveći problem u praktičnoj realizaciji Smitove metode automatske regulacije je izrada kompenzatora relativno dugog transportnog kašnjenja τ. Često je potrebno da taj kompenzator ima mogućnost jednostavne promene veličine τ - neki put u širokim granicama ukoliko se traži da je kompenzator univerzalan. Jer, na primer, u hemijskom inženjerstvu, postoje procesi kod kojih kašnjenje iznosi nekoliko sekundi ali i procesi sa kašnjenjem od jednog sata.

Do sada je razvijeno više načina simulacije transportnog kašnjenja. Najpoznatiji način koristi analogne računske elemente i Padé - ovu aproksimaciju transportnog kašnjenja. Na bazi toga je izradjen i Smitov kompenzator /2/. Mane su mu: u izlaznom signalu se javljaju parazitne oscilacije, kašnjenja duža od jednog minuta se ne mogu realizovati i nedovoljna tačnost. Osim realizacije pomoću analognih računarskih elemenata postoje i simulatori transportnih kašnjenja izradjeni od nekih digitalnih elektronskih elemenata (napr. od pomeračkih registara) ili neelektričnih komponenti (napr. fluidni simulator). Zajednički nedostatak ovih simulatora je što im nedostaje fleksibilnost - veličinu kašnjenja nije moguće menjati ni ručno niti automatski u širokom opsegu.



Sl. 2. Blok-šema simulatora kašnjenja



Sl. 3. Blok-šema regulacije ekstrakcione kolone

OSOBINE I VERZIJE SIMULATORA

Zbog svega izloženog je razvijen mikroračunarski simulator (ručno) promenljivog transportnog kašnjenja - μRS, koji je kompatibilan, tj. može se direktno spregnuti na analognim računskim elementima. To je neophodno kako za simulaciju i ispitivanje procesa (kada se proces predstavlja analognim simulacionim modelom) tako i u projektovanju sistema automatske regulacije jer je Smitova metoda u suštini kontinualna metoda automatske regulacije dinamičkih sistema iako se zbog mikroračunara uvodi diskretizacija signala. Naime, prilikom projektovanja i sinteze Smitovog regulatora za odredjene dinamičke sisteme, tj. procese, oba transportna kašnjenja (procesa i kompenzatora) se simuliraju pomoću mikroračunarskih simulatora, dok se ostali elementi procesa i regulatora simuliraju pomoću računskih elemenata analognog računara. Osim u sintezi Smitove metode automatske regulacije mikroračunarski simulator μRS može da se koristi kao simulacioni model i u sledećim slučajevima kod procesa sa transportnim kašnjenjem:u parametarskoj identifikaciji, u projektovanju i podešavanju konvencionalnih analognih regulatora sistema sa povratnom spregom ili sistema sa predregulacijom kao i za sisteme adaptivnog upravljanja.

Blok-šema mikroračunarskog simulatora promenljivog transportnog kašnjenja na bazi mikroprocesora MC6800 prikazana je na sl. 2, a njegove glavne karakteristike su:

- ulazno-izlazni opseg analognog signala: 0 do 10v (ili -5 do +5v),
- ručno-promenljivo kašnjenje: τ = 1 do 6 sec,
- vreme rada (tj. posmatranje pojave): Tp = τ + 28 sec.

U slučaju transportnog kašnjenja τ reda sekundi tajmer na sl. 2 je običan monostabilni multivibrator sa promenljivim otpornikom R. Radni program simulatora, smešten u EPROM-u, se sastoji od dve rutine (restart i interapt rutine) i detaljno je prezentiran u referenci /3/. Ako je potrebna dužina transportnog kašnjenja reda minuta ili desetine minuta (na primer, radi izrade realnog Smitovog regulatora ili radi kompatibilnosti sa pneumatskim simulatorom procesa) ulogu tajmera preuzima programirajući tajmer MC6840 (Motorola) koji omogućava tako duga kašnjenja. Za to su potrebne i neke izmene u softveru - izmedju ostalog treba povećati periodu odmeravanja (koja se generiše malim potprogramom) analogno-digitalnog konvertora. Ovakav simulator se, uz dodatnu malu izmenu radnog programa, može modificirati tako da se umesto izradjenog simulatora za laboratorijski rad (sa ograničenim vremenom posmatranja) može dobiti simulator za rad u realnom vremenu, tj. za kontinualan rad u pogonskim uslovima procesa. Takav simulator se može koristiti kao kompenzator u pravom Smitovog regulatora i za još neke primene. Ukoliko transportno kašnjenje procesa nije tačno odredjeno već su u pogonu potrebna izvesna on-line ručna repodešavanja kašnjenja kompenzatora tada se signal za odmeravanje (umesto potprogramom) generiše eksternim oscilatorom; perioda ovog signala linearno zavisi od veličine jednosmernog kontrolnog napona koji se može ručno ili automatski menjati. S druge strane se, pak, lako može programski postići da kašnjenje linearno zavisi od perioda odmeravanja a time i od kontrolnog napona. Verzija simulatora sa automatski promenljivim kašnjenjem bi bila potrebna u slučajevima kada je kašnjenje procesa vremenski promenljivo - na primer ako ono nastaje usled kretanja fluida kroz cevovode a brzina fluida je vremenski promenljiva.

PRIMER PRIMENE SIMULATORA TRANSPORTNOG KAŠNJENJA

Jedna fabrika za proizvodnju sirćetne kiseline /4/ se sastoji od kolone za razdvajanje (ili ekstrakcione kolone) i niza destilacionih kolona. Zadatak (cilj) celog postrojenja je dobijanje visokokoncentrovane (99,9%) sirćetne kiseline od razblažene (20,) sirćetne kiseline; ona se kao sirovina dovodi u vrh ekstrakcione kolone na čijem se izlazu (na dnu) dobija delimično prečišćena sirćetna kiselina - rafinat R - koja se zatim dalje vodi u glavnu destilacionu kolonu tzv. dehidracionu kolonu. Koncentracija kiselina rafinata X_r mora da se održava na optimalnoj, konstantnoj vrednosti u cilju minimalne potrošnje (od strane dehidracione kolone) količine vodene pare koja predstavlja glavni deo ukupnih troškova proizvodnje. Medjutim, koncentracija i protok sirovine se znatno menjaju što ustvari predstavlja poremećaj D u sistemu. Te promene se kompenziraju dotokom rastvarača S u ekstrakcionu kolonu. Ovaj rastvarač recirkuliše a njegov dotok u ekstrakcionu kolonu treba da je doziran, tj. da se menja samo prema potrebi. Zato on treba da bude manipulativna promenljiva u sistemu automatske regulacije izlazne koncentracije X_r s obzirom da postoji i dejstvo poremećaja. Ovo se može predstaviti u domenu kompleksne frekvencije blok-šemom na slici 3.

U referenci /4/ je opisana kompleksna automatska regulacija celog procesa pomoću digitalnog procesnog računara. U ovom radu se, pak, predlaže automatska regulacija hemijskog sastava rafinata ekstrakcione kolone pomoću kontinualnog proporcionalnog regulatora (faktor pojačanja je $K_c = 10$) sa simulatorom transportnog kašnjenja (Smitov princip) koji se realizuje pomoću mikroračunarskog simulatora.

Prenosna funkcija $W_d(s)$ nije poznata dok se na osnovu analize realnog procesa dobija da prenosna funkcija $W_p(s)$ ima sledeći oblik:

$$W_p(s) = \frac{X_r(s)}{S(s)} = \frac{A_p \cdot e^{-Ls}}{(T_1 s + 1)(T_2 s + 1)}$$

Na osnovu dimenzija ekstrakcione kolone i ostalih karakteristika usvaja se da je:

$A_p = 1$, $T_1 = 12$ min. $T_2 = 5$ min.

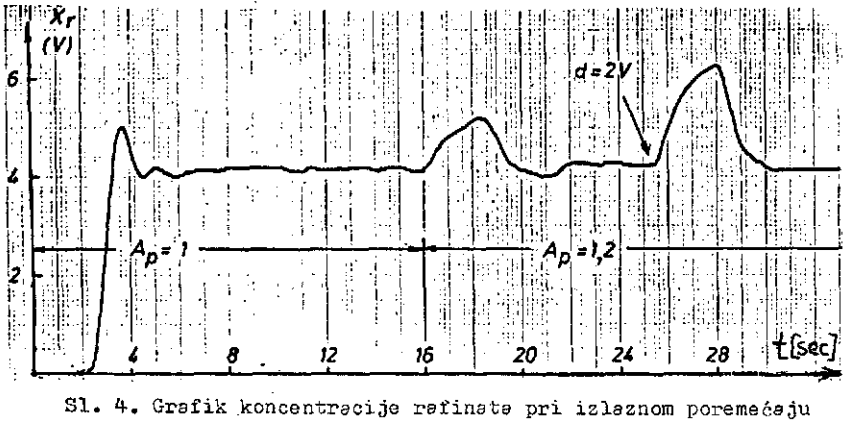
Pošto se hemijski sastav rafinata X_r na izlazu ekstrakcione kolone meri svakih 20 min. pomoću direktno uključenog procesnog analizatora, to će u sistemu automatske regulacije vremenska konstanta transportnog kašnjenja da ima vrednost L = 2o min. Promenljiva D(s) u obliku otskočne funkcije deluje preko svog dinamičkog bloka $W_d(s)$ kao izlazni poremećaj sistema automatske regulacije.

Za simulaciju procesa i regulatora se koristi analogni računar TARA-50 u sprezi sa mikroračunarskim simulatorima koji se koriste za simulaciju transportnih kašnjenja procesa i Smitovog kompenzatora. S obzirom da je realni proces suviše spor prilikom simulacije se uvodi takav faktor vremenske razmere da 1 sec simulacionog modela odgovara 10 min realnog procesa. Dobijeni rezultati pokazuju da se Smitovom metodom lako kompenzira deterministički izlazni poremećaj u obliku otskočne funkcije. Vreme uspostavljanja ponovnog stacionarnog stanja zavisi od dinamike poremećaja (tj. G_d).

U cilju ispitivanja uticaja 'on-line' promene parametara procesa, posle uspostavljanja stacionarnog stanja je izvršena ručna promena vrednosti jednog potenciometra analognog računara tako da to odgovara povećanju (ili smanjenju) faktora pojačanja procesa A_p za 20-25%. Posle kratkog vremena ponovo je uspostavljeno stacionarno stanje. Zatim je (sa tako promenjenim faktorom pojačanja procesa u simulacioni model uključen signal poremećaja d = 2 V na izlazu iz procesa. I pored nejednakosti faktora pojačanja procesa i kompenzatora ($A_p \neq A_m$) ubrzo je ponovo uspostavljeno stacionarno stanje što se vidi iz grafika na slici 4. Zatim je, uz $A_p \neq A_m$, umesto poremećaja d promenjena vrednost signala postavne tačke za $\Delta X_r^o = 2$ V. Nova stacionarna vrednost izlaznog signala je vrlo brzo uspostavljena. Slični rezultati se dobijaju ako se pri nejednakosti $L \neq \tau$ uključuje signal poremećaja d, ili se menja vrednost signala postavne tačke sa $\Delta X_r^o = \pm 2$ V. U oba slučaja relativno brzo se uspostavlja stacionarno stanje.

Svi dobijeni rezultati pokazuju da je ispitivani sistem automatske regulacije stabilan čak iako postoji izvesna nejednakost parametara kompenzatora i procesa. To je u suprotnosti sa nekim ranije dobijenim rezultatima na osnovu ispitivanja /2, 1/ koja su pokazala veliku osetljivost Smitove metode na nepodešenost parametara kompenzatora i procesa ako se radi o relativno dugom transportnom kašnjenju procesa. Medjutim ta ispitivanja (i njihovi zaključci) se ne mogu prihvatiti kao pouzdana jer su kod njih za realizaciju transportnog kašnjenja kompenzatora korišćeni Padé-ova aproksimacija i analogni računski elementi.

Zaključak ovog rada je da je Smitova metoda pogodna za automatsku regulaciju procesa sa dugim transportnim (čistim) kašnjenjem čak i u slučajevima manjih nepodešenosti parametara kompenzatora i procesa - bilo zbog nepoznavanja apsolutno tačnog matematičkog modela procesa ili zbog malih varijacija parametara procesa tokom vremena.



Sl. 4. Grafik koncentracije rafinata pri izlaznom poremećaju

LITERATURA

1. J.E.Marshall:"Control of Time-delay Systems",England,1979.

2. A.Singh,D.H.McEwen:"The control of a process having appreciable transport lag - a laboratory study", IEEE Trans.vol. IECI, No 3, 1975.

3. D.Stojić:"Microprocessor-based simulator of a variable time delay", Microprocessors and microsystems, No 2, 1982.

4. M.Matsushita, T.Shimonaru: "Computer control and computer aided operation of acetic acid recovery process", Symp.on computer control in process industry, Montreux, Suisse, 1979.

PROGRAMSKI REALIZOVANO NALAŽENJE SLOŽENOSTI BULOVE FUNKCIJE

Mančić Vesna
Institut za primenjenu matematiku i elektroniku - Beograd

UDK: 519.852

U radu je prikazan algoritam za nalaženje složenosti Bulove funkcije od n argumenata $(x_1,\ldots,x_n)\in GF(2^n)$ *na bazi njenog kanoničkog polinoma. Funkcija data tabelarno transformiše se algoritmom u kanonički polinom. Za slučaj kada su argumenti funkcije n binarnih periodičnih sekvenci čije su složenosti dve po dve uzajamno proste nalazi se složenost funkcije, direktno na osnovu njenog kanoničkog polinoma.*

COMPUTING THE COMPLEXITY OF BOOLEAN FUNCTION: An algorithm for computing the complexity of Boolean function of n argumente $(x_1,\ldots,x_n)\in GF(2^n)$ *on the basis of canonical polynomial form is presented. The algorithm transforms the function given in any form into canonical polynomial form. In the case when the arguments are n periodical binary sequences whose complexities are in pairs relatively prime, complexity of the function is computed directly out of the canonical polinomial form.*

1. UVOD

Linearni pomerački registar (LPR) sa r stanja može se koristiti za generisanje periodične binarne sekvence $[s(t)]$ koja zadovoljava rekurentnu relaciju oblika

$$s_r=\sum_{i=0}^{r-1}c_i s_{r-i}\ ,\ c_i\in\{0,1\}\ ,\ c_r=1,\ t=0,1,2,\ldots \quad (1)$$

gde je aritmetika u GF(2). Polinom

$$c(y)=1+c_1y+c_2y^2+\ldots+c_{r-1}y^{r-1}+y^r \quad (2)$$

stepena r sa koeficijentima u GF(2) je generišući polinom sekvence s (t) i šift registra koji produkuje, videti /4/. Ako je c(y) nesvodljiv linearna složenost je stepen polinoma inače je $\leqslant r$. Značaj linearne složenosti C je u tome da ako je poznato 2C uzastopnih elemenata sekvence moguće je izvršiti linearnu rekonstrukciju celokupne sekvence.

U ovom radu tretiraćemo proizvoljne Bulove funcije od n argumenata sa ciljem da nadjemo njihovu složenost. Argumenti su binarne periodične sekvence i svaka od njih ima svoj linearni ekvivalent, tj. LPR minimalne dužine koji je može generisati. Ako je funkcija data tabelarno treba da se svede na kanonički polinom, algoritmom koji ćemo prikazati u delu 2., a onda u delu 3. pomoću analitički odredjene linearne složenosti na osnovu kanoničkog polinoma opisaćemo program za nalaženje linearne složenosti Bulove funkcije, sa primerima.

2. VEZA IZMEDJU SDNF I KANONIČKOG POLINOMA

Poznato je da se Bulova funkcija od n promenljivih može predstaviti različitim formama. Od svih tih formi veoma je važna savršena disjunktivna normalna forma (SDNF), jer je jedinstvena i svi drugi oblici logičke funkcije mogu se svesti na nju, (dokaz npr. u /5/). Takodje se svaka Bulova funkcija može predstaviti kanoničkim polinomom, samo pomoću sabiranja po mod2 i konjukcije.

Posmatramo bilo koju logičku funkciju koja preslikava $GF(2^n)\longrightarrow GF(2)$ i $x=(x_1,\ldots,x_n)$ i $a=(a_1,\ldots,a_n)$ pripadaju $GF(2^n)$.

SDNF je oblika: $f(x_1,\ldots,x_n)=\bigvee_{a\in GF(2^n)}\tilde{x}_1\tilde{x}_2\ldots\tilde{x}_n$

i gde je $\tilde{x}_i=\begin{cases}x_i, & a_i=1\\ \bar{x}_i, & a_{i=0}\end{cases}\quad i=\overline{1,n}$ (3)

gde su $\tilde{x}_1\tilde{x}_2\ldots\tilde{x}_n$ potpuni proizvodi koji odgovaraju svim vektorima na kojima funkcija ima vrednost 1.

Označimo sa

$$x^a = x_1^{a_1} \cdot x_2^{a_2} \cdot \ldots \cdot x_n^{a_n} \quad \text{gde je} \quad x_i^{a_i} = \begin{cases} x_i & \text{za } a_i = 1 \\ 1 & \text{za } a_i = 0 \end{cases}$$

Kanonički polinom proizvoljne logičke funkcije je:

$$f(x) = \sum_{a \in GF(2^n)} A_a x^a, \qquad A_a \in \{0,1\} \tag{4}$$

Ako je funkcija f data u proizvoljnom obliku ona se može svesti na SNDF, a znajući nju možemo odrediti koeficijente A_a na sledeći način, vidi /1/.

Definišemo parcijalno uredjivanje $GF(2^n)$ kao $x \leqslant y$, ako i samo ako je $x_j \leqslant y_j$, $1 \leqslant j \leqslant n$.

Neka je $I_a = \{x : x \leqslant a\}$. Tada važi sledeća lema:

$$\sum_{x \in I_b} x^a = \begin{cases} 1 & \text{za } b = a \\ 0 & \text{za } b \neq a \end{cases}$$

Njena direktna posledica glasi:

Ako je $f(x) = \sum_a A_a x^a$ tada je $A_a = \sum_{x \in I_a} f(x)$ (5)

Kanonički polinom, napisan u razvijenom obliku, funcije f:

$$f(x_1, x_2, \ldots, x_n) = A_{a_0} + A_{a_1} x_1 + \ldots + A_{a_n} x_n +$$

$$+ A_{a_{n+1}} x_1 x_2 + A_{a_{n+2}} x_1 x_3 + \ldots + A_{a_{2^n-1}} x_1 x_2 \ldots x_n \tag{6}$$

gde a_i označava n-torku nula i jedinica koja odgovara binarnom zapisu broja i. Npr: i=7, $a_7 = (0,0,\ldots,0,1,1,1)$.

Postoje i drugi načini da se dobije kanonički polinom ako je poznata SDNF funkcija, ali kao što ćemo videti u 3. ovaj je efikasno programski simuliran na računaru i za veliki broj promenljivih.

3. VEZA IZMEDJU KANONIČKOG POLINOMA BULOVE FUNKCIJE I NJENE SLOŽENOSTI

Posmatrajmo Bulovu funkciju od n promenljivih

$$f(s),\ s = (s_1(t), \ldots, s_n(t)),\ t = 0,1,2,\ldots \tag{7}$$

čiji su argumenti periodične binarne sekvence. Za jednu n-torku dobije se kao vrednost funkcije f samo jedna vrednost iz $\{0,1\}$ tako da za t= 0,1, 2,... dobijemo sekvencu $[f(s(t))] = [r]$ čiju složenost želimo da odredimo. Tu složenost kraće zovemo složenost funkcije.

Pošto je to periodična binarna sekvenca ona može biti generisana familijom LPR-a. Postoji jedinstven član te familije koji je najkraći od svih i najbolji način da se on odredi opisao je Massey u /3/. Njegovu programsku realizaciju, iz /2/, smo koristili da bi odredili složenosti proizvoljnih funkcija. Analize smo vršili više puta za različite funkcije i zaključili odredjene pravilnosti. Gornja granica složenosti Bulove funkcije jednaka je vrednosti koja se dobije kada se u kanoničkom polinomu izrazi za argumente zamene odgovarajućim složenostima elementarnih sekvenci. Kada funkcija f, data sa (4), ima za argumente sekvence čije su linearne složenosti C_i, $i=1,2,\ldots,n$, po parovima uzajamno proste, onda važi teorema dokazana u /6/.

Teorema: Linearna složenost izlazne sekvence [r] definisane preko (7) odredjena je relacijom:

$$C = \sum_{a \in GF(2^n)} A_a \prod_{i=1}^{n} C_i^{a_i} \tag{8}$$

i	x_1	x_2	x_3	$f(x_1\ x_2\ x_3)$
0	0	0	0	1
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	0
7	1	1	1	0

Slika br. 1

Pretpostavimo da su argumenti funkcije, tabelarno date (sl.1), nizovi uzajamno prostih složenosti.

Kada simuliramo funkciju $f(x_1, x_2, x_3)$ i dobijeni izlazni niz odredjene dužine, analiziramo Massey-evim algoritmom, dobijamo da je složenost niza vrednosti funkcije jednaka 29. Da bi potvrdili upravo dobijeni rezultat pristupamo sledećoj proceduri:

1) Nalazimo pomoću tabele SDNF date funkcije

SDNF $f(x_1,x_2,x_3) = \bar{x}_1\bar{x}_2\bar{x}_3 \vee \bar{x}_1\bar{x}_2x_3 \vee \bar{x}_1x_2x_3 \vee x_1\bar{x}_2\bar{x}_3$

2) Nalazimo kanonički polinom date funkcije. Na osnovu (5):

i	I_a	$A_a = \sum_{x \in I_a} f(x)$	
0	{000}	f(0)	1
1	{000,001}	f(0)+f(1)	0
2	{000,010}	f(0)+f(2)	1
3	{000,001,010,011}	f(0)+f(1)+f(2)+f(3)	1
4	{000,100}	f(0)+f(4)	0
5	{000,001,100,101}	f(0)+f(1)+f(4)+f(5)	1
6	{000,010,100,110}	f(0)+f(2)+f(4)+f(6)	0
7	$GF(2^3)$	f(0)+f(1)+...+f(7)	0

Koeficijenti kanoničkog polinoma A_a su 10110100 pa kanonički polinom ima oblik:

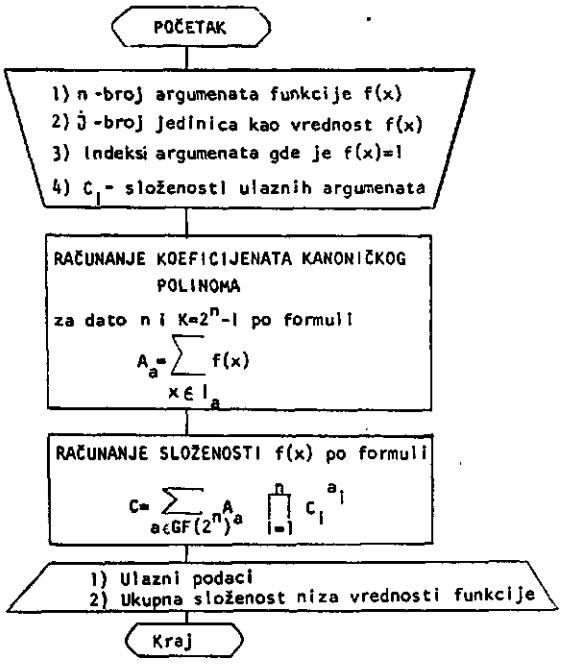
$$f(x_1,x_2,x_3)=1+x_2+x_2x_3+x_1x_3 \qquad (9)$$

Ako prema teoremi umesto x_i stavimo C_i, i=1,2,3 dobijemo: C=(F(x)=1+3+3.5+2.5=29

ALGORITAM ZA NALAŽENJE SLOŽENOSTI

U prethodnom primeru izložili smo proceduru za nalaženje koeficijenata kanoničkog polinoma, a samim tim i potvrdili formulu za nalaženje složenosti izlaznog niza funkcije, za n=3. Simulirali smo algoritam za različite vrednosti n.

Program **za nalaženje složenosti** je jednostavan i brz, a njegova blok šema je sledeća:



ZAKLJUČAK

Posmatrali smo Bulovu funkciju čiji su argumenti binarne periodične sekvence, a složenosti uzajamno proste po parovima. Za takve argumente važi teorema za nalaženje složenosti izlazne funkcije, pri čemu se pod složenošću podrazumeva dužina minimalnog pomeračkog registra koji može da generiše identičnu binarnu sekvencu. Složenost se direktno računa na osnovu kanoničkog polinoma funkcije.

Da bi se simuliralo nalaženje složenosti proizvoljne Bulove funkcije bilo je potrebno simulirati i nalaženje koeficijenta kanoničkog polinoma. Napisan je algoritam koji za tabelarno datu funkciju čiji argumenti zadovoljavaju uslove teoreme nalazi koeficijente kanoničkog polinoma i na osnovu njih složenost date funkcije.

LITERATURA

/1/ Tore Herles tam:"On functions of linear shift register sequences",septembar 1981. god. submitted to IEEE Trans. Inform. Theory.

/2/ Golić Jovan:"Primena nekih metoda teorije informacija u analizi i sintezi sekvenci simbola sa aspekta generisanja pomoću linearnih pomeračkih registara" - magistarski rad, Beograd.

/3/ J. L. Massey:"Shift-register synthesis and BCHdecoding", IEEE Trans. Information Theory, VOL. IT-15. January 1969. pp122-127.

/4/ Solomon W. Golomb:"Shift Register Sequences" San Francisco,Holden-Day,1967. god.

/5/ Borivoj Lazić:"Logičko projektovanje I", ETF u Beogradu, 1976. godine.

/6/ Jovan Golić, Miodrag Mihaljević:"Analiza minimalnog linearnog ekvivalenta jedne klase binarnih automata", ETAN u pomorstvu, Zadar 1985. godine.

P R I L O G : REZULTATI PROGRAMA

PRIMER BR. 1

PROGRAM: NOVA GORICA

```
16 VREDNOSTI F-JE OD  4 PROMENLJIVE
 0 1 1 0 1 0 1 1 1 1 1 0 0 0 0 0
KOEFICIJENTI KANONICKOG POLINOMA
 0 1 1 0 1 0 1 1 1 1 1 1 0 0 1 0

ELEMENTARNI PROIZVODI KOJI FIGURISU
U KANONICKOM POLINOMU
   x1x2 x3 x4
   0 0 0 1
   0 0 1 0
   0 1 0 0
   0 1 1 0
   0 1 1 1
   1 0 0 0
   1 0 0 1
   1 0 1 0
   1 0 1 1
   1 1 1 0

SLOZENOSTI ULAZNIH NIZOVA PO PAROVIMA
UZAJAMNO PROSTE
    C1      C2      C3      C4

     7      11      17      37
UKUPNA SLOZENOST IZLAZNOG NIZA JE   13268
```

PRIMER BR. 2

PROGRAM: NOVA GORICA

```
4 VREDNOSTI F-JE OD  2 PROMENLJIVE
 1 1 0 1
KOEFICIJENTI KANONICKOG POLINOMA
 1 0 1 1

ELEMENTARNI PROIZVODI KOJI FIGURISU
U KANONICKOM POLINOMU

   0 0
   1 0
   1 1

SLOZENOSTI ULAZNIH NIZOVA PO PAROVIMA
UZAJAMNO PROSTE
   C1      C2

   11      19
UKUPNA SLOZENOST IZLAZNOG NIZA JE      221
```

PRIMER BR. 3

PROGRAM: NOVA GORICA

```
NIZ VREDNOSTI F-JE F OD  5 PROMENLJIVIH IZ TABELE
 0 0 0 1 0 0 1 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0
NIZ KOEFICIJENATA KANONICKOG POLINOMA
 0 0 0 1 0 0 1 0 0 1 0 0 0 1 1 0 0 0 0 0 0 0 1 1 0 1 0 1 1 1 0 0

ARGUMENTI ZA KOJE SU KOEFICIJENTI KANONICKOG POLINOMA JEDNAKI 1
   x1 x2 x3 x4 x5
   0 0 0 1 1
   0 0 1 1 0
   0 1 0 0 1
   0 1 1 0 1
   0 1 1 1 0
   1 0 1 1 0
   1 0 1 1 1
   1 1 0 0 1
   1 1 0 1 1
   1 1 1 0 0
   1 1 1 0 1

SLOZENOSTI ULAZNIH NIZOVA
   C1      C2      C3      C4      C5

    2       3       5       7      11
UKUPNA SLOZENOST IZLAZNOG NIZA JE          2143.
```

PRIMER BR. 4

PROGRAM: NOVA GORICA

```
 8 VREDNOSTI F-JE OD  3 PROMENLJIVE
 1 1 0 1 0 1 1 0
KOEFICIJENTI KANONICKOG POLINOMA
 1 0 1 1 1 1 0 1

ELEMENTARNI PROIZVODI KOJI FIGURISU
U KANONICKOM POLINOMU

   0 0 0
   0 1 0
   0 1 1
   1 0 0
   1 0 1
   1 1 1

SLOZENOSTI ULAZNIH NIZOVA PO PAROVIMA
UZAJAMNO PROSTE

 C1      C2      C3

  2       3       5
UKUPNA SLOZENOST IZLAZNOG NIZA JE       61
```

ALGORITMI ZA REINVERZIJO BAZNE MATRIKE V LINEARNEM PROGRAMU

Janez Barle, Janez Grad
Univerza Edvarda Kardelja v Ljubljani,
Ekonomska fakulteta Borisa Kidriča, Ljubljana

UDK: 519.852.11

V prispevku opišemo nekatere algoritme za invertiranje matrik, ki so uporabni pri reševanju linearnih programov. Njihova značilnost je specifičen način predstavitve inverzne matrike in razdelitev postopka v dve fazi. Poseben poudarek dajemo problemom, ki se nanašajo na računalniško realizacijo algoritmov.

BASIS MATRIX REINVERSION ALGORITHMS IN LINEAR PROGRAMMING. In this paper we describe some of the algorithms for matrix reinversion which can be applied in the procedures for solving the linear programming problem. They use a specific presentation of the inverse matrix and two-phase computation process. Special attention is paid to those problems associated with the computer processing of these algorithms.

UVOD

Reševanje sistema linearnih enačb in s tem povezan problem invertiranja matrike spadata med klasične probleme numerične linearne algebre. Običajne metode za reševanje linearnih enačb uporabljajo Gaussov eliminacijski postopek, pri katerem se zaradi zagotavljanja numerične stabilnosti izvaja t.i. pivotiranje (delno ali popolno). Pri delu na programski opremi za linearno programiranje smo prišli do spoznanja, da omenjene metode niso vedno učinkovite. Zaradi lastnosti matrik, ki se pojavljajo pri praktičnih problemih linearnega programiranja, je treba za njihovo invertiranje uporabiti posebne postopke. Uporaba teh postopkov lahko precej poveča hitrost in natančnost pri reševanju linearnih programov velikih dimenzij. Zato je podprogram za invertiranje matrike ena od najbolj pomembnih sestavin komercialnih programskih paketov za linearno programiranje. V nadaljevanju podajamo opis nekaterih najbolj znanih algoritmov za invertiranje matrike pri linearnem programiranju. Opisujemo tudi nekatere izkušnje, ki smo jih pridobili pri programiranju in testiranju teh algoritmov.

Problem linearnega programiranja lahko v matrični obliki opišemo takole:

$$\text{minimiziraj } c^T x$$

$$\text{pri pogojih}$$

$$Ax = b \text{ in } x \geq 0$$

kjer je A realna matrika dimenzije m*n, b, c in x pa realni vektorji ustreznih dimenzij. Optimalna rešitev linearnega programa je tak vektor x, ki reši gornji problem.

Upoštevati moramo, da pri večini praktično pomembnih problemov linearnega programiranja velja:

1) Matrika A je zelo velikih dimenzij (tudi do nekaj tisoč vrstic in stolpcev)

2) Matrika A je razpršena. To pomeni, da je delež njenih neničelnih elementov zelo majhen (pogosto tudi manj od 1%). Zaradi tega se v praksi za reševanje linearnih programov skoraj vedno uporablja t.i. revidirana metoda simpleksov, ki jo je možno prilagoditi delu z matrikami velikih dimenzij. To je iterativna metoda, ki je podrobno opisana v literaturi (npr. [1]). Omenimo le, da se na vsakem koraku te metode rešuje zaporedje sistemov linearnih enačb

$$\pi B = c_B,\ Bv = u \text{ in } Bx_B = b$$

kjer je B nesingularna kvadratna podmatrika matrike A (imenujemo jo bazna matrika), c_B in x_B sta ustrezna podvektorja vektorjev c in b, u pa je stolpec matrike A, ki je določen z rešitvijo prvega sistema enačb. Reševanje gornjih sistemov enačb olajšuje dejstvo, da se bazni matriki pri dveh zaporednih korakih algoritma razlikujeta le v enem stolpcu.

PREDSTAVITEV INVERZNE BAZNE MATRIKE

Revidirana metoda simpleksov ima več inačic, ki izhajajo iz različnih načinov predstavitve matrike B (ali B^{-1}). Predstavitev matrike B^{-1} v eksplicitni obliki lahko uporabimo le pri problemih majhnih dimenzij. V obstoječih programskih paketih je najbolj običajna predstavitev inverzne bazne matrike v obliki produkta t.i. elementarnih matrik:

$$B^{-1} = E_k E_{k-1} \ldots E_1$$

Elementarne matrike E_i (i = 1, ... k) se razlikujejo od enotne matrike le v enem stolpcu, tako da zadostuje shraniti neničelne elemente tega stolpca in podatke o tem, kje se le-ti nahajajo. Ena od prednosti produktne predstavitve inverzne bazne matrike je tudi enostaven način njenega ažuriranja. Po vsakem koraku revidirane metode simpleksov se v produkt dodaja nova elementarna matrika, stare elementarne

matrike pa ostanejo nespremenjene. Tako predstavitev matrike imenujemo tudi ETA-datoteka, tisti stolpec, v katerem se elementarna matrika razlikuje od enotne pa imenujemo ETA-vektor.

Znano je, da pri reševanju linearnih enačb po Gaussovem eliminacijskem postopku računanje inverzne matrike ni potrebno. Boljša pot za reševanje sistema enačb (Bx = b) je izvršiti razcep permutirane matrike na spodnjo in zgornjo trikotno matriko (PB = LU) in potem reševati dva trikotna sistema enačb (Ly = b in Ux = y). Ta ugotovitev v določeni meri velja tudi za bazno matriko linearnega programa. Za njo lahko uporabimo razcep B = LU, kjer vsakega od obeh faktorjev predstavimo z zaporedjem ustreznih trikotnih elementarnih matrik:

$L = L_1 L_2 \ldots L_k$ in $U = U_k \ldots U_1$

Tako predstavitev imenujemo eliminacijska oblika inverzne matrike. V zapisu smo zaradi poenostavitve izpustili permutacijske matrike, ki so podane z zaporedjem, v katerem obravnavamo vrstice in stolpce matrike B. Izkaže se, da v splošnem uporaba eliminacijske oblike matrike omogoča ekonomičnejšo izrabo pomnilnika in časovno hitrejši izračun rešitve linearnega programa v primerjavi s postopkom, ki uporablja produktno obliko inverzne matrike. Vseeno so trikotni razcep začeli vgrajevati v komercialne programske pakete šele potem, ko so bili razviti učinkoviti postopki za ažuriranje trikotnih faktorjev (npr. metoda Forresta in Tomlina, glej |1|). Ugotavljamo pa, da uporaba trikotnega razcepa zahteva precej bolj zapletene algoritme in podatkovne strukture kot postopek, ki uporablja produktno obliko inverzne matrike.

REINVERZIJA BAZNE MATRIKE

Ker se med izvajanjem simpleksnega algoritma število elementarnih matrik v predstavitvi B^{-1} ali v trikotnih faktorjih L in U povečuje, se izkaže za nujno občasno izračunati novo predstavitev B^{-1} ali novi trikotni razcep matrike B. Ta postopek, imenujemo ga reinverzija bazne matrike, sprošča pomnilnik in omogoča hitrejše in natančnejše izvajanje algoritma na preostalih korakih revidirane metode simpleksov. Naziv reinverzija je primeren tudi za postopek določanja novih elementarnih matrik trikotnega razcepa matrike, ker je ta naloga v tesni povezavi z invertiranjem matrike. Tako lahko iz znane eliminacijske oblike matrike takoj dobimo produktno obliko inverzne matrike:

$$B^{-1} = U^{-1}L^{-1} = U_1^{-1} \ldots U_k^{-1} L_k^{-1} \ldots L_1^{-1}$$

Za produktno in eliminacijsko obliko matrike uporabljamo tudi skupen naziv ETA-faktorizacija matrike. Omenimo še, da je inverzum elementarne matrike zopet elementarna matrika. Vzemimo, da se prvotna elementarna matrika razlikuje od enotne matrike v stolpcu, ki ga označimo z y. Pri inverzumu se ta stolpec spremeni po naslednjih pravilih: diagonalni element y_k postane $1/y_k$, vsi ostali elementi y_i pa se spremenijo v $-y_i/y_k$.

Pri reinverziji izhajamo iz prvotne matrike B, ki je praviloma razpršena. Lahko se zgodi, da ima matrika B^{-1} mnogo več neničelnih elementov kot B. To velja tudi za skupno število neničelnih elementov v ETA-faktorizaciji inverzne matrike. Ta pojav, imenujemo ga napolnitev, je posebno izrazit takrat, ko pri izvajanju Gaussovega eliminacijskega postopka uporabimo delno ali popolno pivotiranje. To sta postopka, pri katerih se teži k izbiri ključnega elementa (pivota) s čim večjo absolutno vrednostjo ("pivoting for size"). Zato je pri reinverziji bazne matrike treba uporabiti take kriterije za izbor pivotov, ki vodijo k ETA-faktorizaciji s čim manjšim številom neničelnih elementov ("pivoting for sparsity"). Poskusi optimalne rešitve tega problema, ki je znan tudi pod imenom "problem minimalne napolnitve", so pripeljali do ugotovitve, da je ta problem NP-poln. To pomeni, da je malo upanja, da bo za njegovo reševanje kdaj oblikovan ekonomičen algoritem (glej |10|). Zato so bili razviti številni algoritmi, osnovani na različnih heurističnih pravilih za izbor pivota, ki so bolj ali manj uspešno reševali zastavljeno nalogo. Pomemben dogodek v raziskovanju tega problema je bil, ko je leta 1957 H. Markowitz objavil članek o tej problematiki (glej |8|). V njem je postavil nekaj trditev, ki so spodbudile številne poznejše raziskave:

1) Postopek invertiranja je smiselno razdeliti v dve fazi. V prvi fazi, imenujemo jo analitična faza, se vnaprej določi zaporedje, po katerem se bo vršilo pivotiranje. Pri tem se uporabljajo le informacije o tem, kateri elementi so neničelni, ne oziraje se na njihovo vrednost. V drugi fazi, imenujemo jo numerična faza, se na podlagi vnaprej določenega zaporedja pivotov izvrši dejanski izračun nove ETA-faktorizacije.

2) Dobre rezultate pri minimizaciji neničelnih elementov v ETA-faktorizaciji lahko dosežemo tako, da na posameznem koraku izbiramo tisti pivot, pri katerem ima "Markowitzevo število"

   $(v_i - 1)(s_j - 1)$

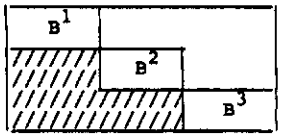
   minimalno vrednost. Z v_i ("vrstični števec") in s_j ("stolpični števec") sta označeni števili neničelnih elementov v i-ti vrstici in j-tem stolpcu, pri čemer upoštevamo le tiste vrstice in stolpce, v katerih še ni bilo izvršeno pivotiranje.

Markowitz je bil tudi prvi, ki je ugotovil primernost eliminacijske oblike za predstavitev bazne matrike pri linearnem programiranju.

ANALITIČNA FAZA REINVERZIJE

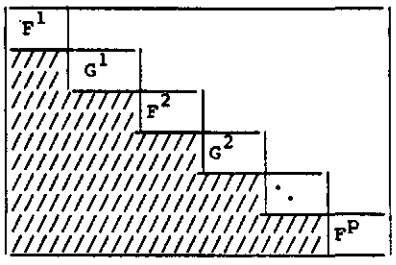
Ideja o ločenem izvajanju analitične faze se ni uveljavila le pri reinverziji bazne matrike, temveč tudi pri vseh drugih problemih numerične matematike, ki zahtevajo reševanje razpršenih sistemov linearnih enačb. Praksa je potrdila tudi to, da uporaba Markowitzevega pravila pri izbiri pivota vodi k zelo razpršeni ETA-faktorizaciji inverzne matrike. Vseeno je direktna uporaba tega pravila otežkočena, ker zahteva hkraten dostop do vrstic in stolpcev matrike. Kolikor nam je znano, uporablja Markowitzevo pravilo v čisti obliki le podprogram za reinverzijo pri IBM-ovem programskem paketu MPSX/370 (glej |2|). Izkazalo se je, da je možno uporabiti različne kriterije, ki na posreden način merijo Markowitzevo število, hkrati pa zahtevajo veliko manj računanja.

Sodobni programski paketi v analitični fazi reinverzije večinoma uporabljajo algoritem Hellermana in Raricka (glej |6| in |7|). Ta algoritem je prirejen strukturi matrik, ki so značilne za praktične probleme linearnega programiranja, s čimer lahko pojasnimo njegovo učinkovitost. Cilj tega algoritma je, da se z zamenjavo vrstic in stolpcev matrika B prevede v t.i. HR matriko, ki ima strukturo kot na sliki 1,



Slika 1

kjer so B^1 in B^3 spodnje trikotne matrike z neničelnimi elementi na diagonali. Lahko privzamemo, da ima B^2, če ni prazna, v vsaki vrstici in stolpcu vsaj dva neničelna elementa, ker bi v nasprotnem primeru lahko razširili B^1 ali B^3. Matriko B^2 imenujemo jedro ali izboklina ("bump"). Algoritem Hellermana in Raricka se nadaljuje z ustreznimi operacijami na jedru, dokler jedro ne postane strukturirano kot na sliki 2, kjer so G^k ali prazne ali spodnje trikotne matrike z neničelnimi elementi na diagonali:



Slika 2

Matrike F^p imenujemo zunanje izbokline ("external bumps"). Imajo vsaj po en stolpec z neničelnimi elementi nad diagonalo. Take stolpce imenujemo konice ("spikes"). Zunanje izbokline se morajo odlikovati z naslednjimi lastnostmi:

i) zadnji stolpec v izboklini je konica z neničelnim elementom v najvišji vrstici,

ii) stolpci, ki niso konice, morajo imeti neničelne diagonalne elemente.

Algoritma ne bomo podrobneje opisovali. Omenimo le, da se konice izbirajo na temelju stolpične številske funkcije $t_k(j)$, kjer

$t_k(j)$ = število neničelnih elementov j-tega stolpca v vrsticah z vrstičnim številom manjšim ali enakim k

Algoritem Hellermana in Raricka pokaže tudi zanimivo povezavo med teorijo grafov in linearnim programiranjem. Binarno matriko, s katero delamo v analitični fazi reinverzije, lahko interpretiramo kot matriko sosednosti usmerjenega grafa. V jeziku teorije grafov pomeni iskanje bločne trikotne oblike matrike (to je oblika matrike na sliki 2) določanje močno povezanih komponent usmerjenega grafa. To je problem, ki je spodbudil precejšnjo pozornost raziskovalcev s področja teorije grafov. Mi smo za reševanje te naloge sprogramirali eno varianto t. i. Tarjanovega algoritma, ki smo jo povzeli po |3|. Zanimivo je, da je uporaba metod iz teorije grafov pripeljala do izboljšane variante algoritma Hellermana in Raricka, ki je v literaturi dobila ime P4 ("Partitioned Preassigned Pivot Procedure"). S to varianto se praviloma dobi večje število zunanjih izboklin in manjše število konic kot s starejšo varianto algoritma, ki je znana pod nazivom P3 ("Preassigned Pivot Procedure").

Na koncu naj omenimo, da so bili na temelju algoritma Hellermana in Raricka razviti še nekateri zanimivi algoritmi, kot je npr. algoritem opisan v |9|.

NUMERIČNA FAZA REINVERZIJE

V numerični fazi reinverzije se, na temelju zaporedja pivotov določenega v analitični fazi, računa nova ETA-faktorizacija inverzne bazne matrike. V tej fazi moramo različno obravnavati produktno in eliminacijsko obliko inverzne matrike, kar ni bilo potrebno v analitični fazi. V |4| in |5| je podan zelo izčrpen pregled različnih algoritmov za izvajanje numerične faze reinverzije. V nadaljevanju podajamo opis algoritma za produktno obliko inverzne bazne matrike, ki smo ga spogramirali v |11|.

Predhodno določeno zaporedje pivotov je podano z vektorjem C, ki vsebuje indekse stolpcev, in z vektorjem R, ki vsebuje indekse vrstic. Razen tega potrebujemo še indekse konic in podatke o tem, kje se začnejo posamezne zunanje izbokline. Zaradi prihranka pomnilniškega prostora je te informacije koristno vključiti v vektorja R in C. To lahko storimo tako, da z negativnim predznakom shranimo v C indekse stolpcev, ki so konice, in v R indekse vrstic, kjer se začnejo posamezne zunanje izbokline.

N1: |Inicializacija|

a) Vpeljemo naslednje parametre:
   (i) i=1 za indeksiranje pivotov,
   (ii) $l=C_i$ za stolpični indeks pivota,
   (iii) $r=|R_i|$ za vrstični indeks pivota,
   (iv) p=1 za indeksiranje začetka zunanje izbokline.

b) Rezerviramo prostor za
   (i) y - realni vektor dimenzije m,
   (ii) ETA-datoteko (predstavitev matrik E_i za $1 \le i \le m$).

c) Definiramo $y = B_{*l}$ (l-ti stolpec matrike B).

N2: |Določanje novega ETA-vektorja|

$$z_k = \begin{cases} 1/y_r & \text{za } k=r \\ y_k & \text{za } k \neq r \end{cases}$$

Vektor z je r-ti stolpec elementarne matrike E_i.

N3: |Testiranje konca algoritma|

a) če je i=m nadaljujemo na N7 (konec algoritma)

b) če je i<m definiramo
   (i) i=i+1
   (ii) $l=|C_i|$
   (iii) $r=|R_i|$

c) če je $R_i<0$ definiramo p=i (začetek nove izbokline)

N4: |Opredelitev konice|

če je $C_i > 0$ definiramo $y = B_{*l}$ in nadaljujemo na N2

N5: |Transformacija konice|

Izračunamo

$y = E_{i-1} \cdots E_p B_{*l}$

(E_p ustreza prvemu stolpcu v izboklini, ki pripada B_{*l})

N6: |Zamenjava konice, če je pivot enak nič|

a) če je $y_r \neq 0$ postopek nadaljujemo na koraku N2

b) Preiščemo vse konice s stolpičnimi indeksi $l = |C_j|$ za j>i, ki se nahajajo

v isti zunanji izboklini, dokler ne najdemo takšne za katero je r-ta komponenta $E_{i-1}\cdots E_k B_{*1}$ neničelna

c) Izračunamo $y = E_{i-1}\cdots E_p B_{*j}$

d) Zamenjamo C_i in C_j

e) Postopek nadaljujemo na koraku N2

N7: |Konec|

Določanje ETA-datoteke je končano.

Kot vidimo, je korak N5 potreben le za tiste stolpce, ki so konice. Edino v teh stolpcih se lahko pri reinverziji poveča število neničelnih elementov. To je razlog, da je algoritem Hellermana in Raricka tako pomemben. Preverjanje ali je pivot enak nič, ki je potrebno na koraku N6, se v praksi zamenjuje z testi, ki zagotavljajo numerično stabilnost postopka.

ZAKLJUČEK

Uporaba matematike v praksi često zahteva poznavanje metod, ki jih ni možno naučiti iz standardnih učbenikov. Ta ugotovitev velja tudi za metode, ki smo jih morali vključiti v podprogram za invertiranje bažne matrike linearnega programa. Naša raziskovalna skupina nadaljuje s spremljanjem tega raziskovalnega področja in razvojem ustreznih rešitev v sklopu programske opreme za linearno programiranje.

LITERATURA

1) Bastian M., Lineare Optimierung grosser Systeme: Compact Inverse Verfahren und Basisfaktoriesirung, Konigstain: Verlag Anton Hain, 1980.

2) Benichou M., J.M. Gauthier, G. Hentges, G. Ribiere, "The Efficient Solution of Large-Scale Linear Programming Problems - Some Algorithmic Techniques and Computational Results", Mathematical Programming, 13, pp. 280-322, 1977.

3) Gustavson F., "Finding the Block Lower Triangular Form of a Sparse Matrix", v: Bunch J.R., D.J. Rose (eds.), Sparse Matrix Computations, New York: Academic Press, 1976.

4) Helgason R.V., J.L. Kennington, "Spike Swapping in Basis Reinversion", Naval Research Logistic Quarterly, 27, pp. 697-701, 1980.

5) Helgason R.V., J.L. Kennington, "A note on Splitting the Bump in an Elimination Factorisation", Naval Research Logistics Quarterly, 29, pp. 169-178, 1982.

6) Hellerman E., D. Rarick, "Reinversion with the Preassigned Pivot Procedure", Mathematical Programming, 1, pp. 195-216, 1971.

7) Hellerman E., D. Rarick, "The Partitioned Preassigned Pivot Procedure", v: Rose D.J., R.A. Willoughby (eds.), Sparse Matrices and Their Applications, New York-London: Plenum Press, 1972.

8) Markowitz H., "The Elimination Form of the Inverse and its Application to Linear Programming", Management Science, 3, pp. 255-269, 1957.

9) Lin T.D., R.S.H. Mah, "Hierarchical Partition - a New Optimal Pivoting Algorithm", Mathematical Programming, 12, pp. 260-278, 1977.

10) Petkovšek M., "NP-polni problemi", v: M. Petkovšek, Pisanski T., Izbrana poglavja iz računalništva 1. del, Ljubljana: DMFA SRS, 1982.

11) Barle J., J. Grad, V. Rupnik, "Programska oprema za rešitev linearnega programa z razširitvami", Raziskovalni projekt po naročilu Iskre-Delta, Ljubljana: RCEF, Ekonomska fakulteta Borisa Kidriča v Ljubljani, 1984.

STEPENOVANJE DINAMIČKE MATRICE

PRIMENOM PODMATRICA

Slobodan B. Furundžić
"Energoprojekt", Beograd

UDK: 519.852

Indukcijom su, na osnovu generalizacije matričnog množenja, formulisani rekurentni obrasci za stepenovanje matrice preko četiri kvadratne podmatrice. Zatim su, kao specijalni slučajevi, izvedeni odgovarajući obrasci za stepenovanje prigušene i neprigušene dinamičke matrice diskretnih sistema, a korišćenje ovih obrazaca je pokazano na brojnom primeru.

DYNAMIC MATRIX POWERING APPLYING SUBMATRICES: A powering of matrix by four square submatrices is firstly considered. The recurrent formulas for powering of dynamic matrix of discrete systems are than derived, and the application of these formulas is shown by a numerical example.

UVOD

Za rešavanje problema prinudnih vibracija linearnih, stacionarnih, diskretnih dinamičkih sistema, nedavno smo predložili /1/ i programirali /2/ jedan numerički postupak, prema kome se rešenje vibracija dobija stepenovanjem dinamičke matrice sistema.

Ako se ovo matrično stepenovanje vrši po definiciji, to jest sukcesivnim množenjem pune dinamičke matrice, utrošak memorije i vremena računara je veći, naročito u slučaju sistema i stepena visokog reda. Kada se, medjutim, dinamička matrica posmatra kao skup od četiri kvadratne podmatrice, tada se pri stepenovanju uočava jedna zakonitost, na osnovu koje se indukcijom lako mogu da izvedu efikasni rekurentni obrasci.

Te obrasce za stepenovanje dinamičke matrice primenom podmatrica, do kojih smo došli razvijajući pomenuti postupak — pa smatramo da su novina, prikazaćemo u ovom kratkom saopštenju.

JEDNA PRIMENA PODMATRICA

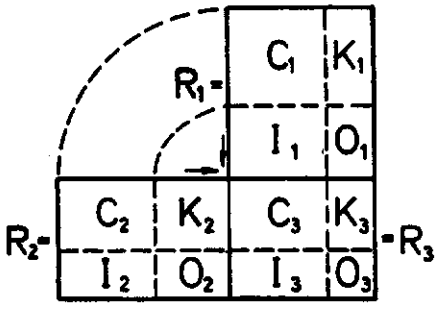
Primena podmatrica, kao što je poznato, često olakšava manipulisanje i operisanje s matricama, a pri korišćenju računara omogućuje uštedu memorije i vremena. Kao jednu mogućnost primene podmatrica, u sledećem, razmotrimo matrično množenje i stepenovanje podelom na četri podmatrice.

Matrično množenje.-- Neka su R_1 i R_2 dve pravougaone matrice, saglasne u odnosu na matrično množenje (sl.1). Kada R_1 podelimo na četiri pravougaone podmatrice $/C_1,K_1,I_1,O_1/$, a zatim R_2 na saglasne podmatrice $/C_2,K_2,I_2,O_2/$, tada u matrici proizvoda:

$$R_3 = R_2 R_1 \qquad (1)$$

saglasno šemi množenja (sl.1), odgovarajuće podmatrice glase:

$$\begin{aligned} C_3 &= C_2C_1 + K_2I_1 \\ K_3 &= C_2K_1 + K_2O_1 \\ I_3 &= I_2C_1 + O_2I_1 \\ O_3 &= I_2K_1 + O_2O_1 \end{aligned} \qquad (2)$$



SL.1 FALKOVA ŠEMA MATRIČNOG MNOŽENJA

Matrično stepenovanje.-- Posmatramo, sada, kvadratnu matricu R^m reda (2n.2n), podeljenu na četiri kvadratne podmatrice $/C_m,K_m,I_m,O_m/$, od kojih je svaka reda (n.n):

$$R^m = \begin{bmatrix} C_m & K_m \\ I_m & O_m \end{bmatrix} \begin{matrix} 1 \\ n \\ 2n \end{matrix}, \quad (m=1,2,\ldots,L) \qquad (3)$$

Kako je po definiciji:

$$R^m = R^{m-1}R^1, \quad (m=2,3,\ldots,L) \qquad (4)$$

to, nakon sukcesivne primene izraza (2), indukcijom dolazimo do sledećih rekurentnih obrazaca za stepenovanje:

$$\begin{aligned} C_m &= C_{m-1}C_1 + K_{m-1}I_1 \\ K_m &= C_{m-1}K_1 + K_{m-1}O_1 \\ I_m &= I_{m-1}C_1 + O_{m-1}I_1 \\ O_m &= I_{m-1}K_1 + O_{m-1}O_1 \end{aligned} \qquad (m=2,3,\ldots,L) \qquad (5)$$

STEPENOVANJE DINAMIČKE MATRICE

U daljem, izvedimo dva specijalna slučaja obrazaca (5). Razmatramo diskretni dinamički sistem sa n stepeni slobode, opisan matricama m, c, k (mase, prigušenja, krutosti), od kojih je svaka reda (n.n).

Prigušena dinamička matrica.-- Ako postoji prigušenje ($c\neq 0$), dinamička matrica sistema R i njen stepen R^m izraženi preko podmatrica glase:

$$R = \left[\begin{array}{c|c} C & K \\ \hline I & O \end{array}\right]; \quad R^m = \left[\begin{array}{c|c} C_m & K_m \\ \hline I_m & O_m \end{array}\right], \quad (m=2,3,\ldots,L) \qquad (6)$$

gde je:

$$\begin{aligned} &C = -m^{-1}c \; ; \quad K = -m^{-1}k \; ; \\ &I - \text{jedinična matrica;} \\ &O - \text{matrica nula.} \end{aligned} \qquad (7)$$

Dalje, iz izraza (5),(6) i (7), neposredno sledi:

$$\begin{aligned} C_m &= C_{m-1}C + K_{m-1} \\ K_m &= K_{m-1}K \\ I_m &= I_{m-1}C + O_{m-1} \\ O_m &= I_{m-1}K \end{aligned} \qquad (m=2,3,\ldots,L) \qquad (8)$$

Kada izraze (8) ispišemo za $m=2,3,\ldots,L$, tada indukcijom za poslednja dva, od ova četri, izraza nalazimo:

$$\begin{aligned} I_{m-1}C + O_{m-1} &= C_{m-1} \\ I_{m-1}K &= K_{m-1} \end{aligned} \qquad (m=2,3,\ldots,L) \qquad (9)$$

Konačno, iz izraza (8) i (9), dobijamo rekurentne obrasce za stepenovanje prigušene dinamičke matrice:

$$\left|\begin{array}{l} C_m = C_{m-1}C + K_{m-1} \\ K_m = K_{m-1}K \\ I_m = C_{m-1} \\ O_m = K_{m-1} \end{array}\right. \qquad (m=2,3,\ldots,L) \qquad (10)$$

Neprigušena dinamička matrica.-- Ako nema prigušenja ($c=0$), dinamička matrica R i njen stepen R^m glase:

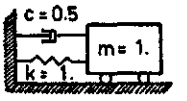
$$R = \left[\begin{array}{c|c} O & K \\ \hline I & O \end{array}\right]; \quad R^m = \left[\begin{array}{c|c} C_m & K_m \\ \hline I_m & O_m \end{array}\right], \quad (m=2,3,\ldots,L) \qquad (11)$$

a iz izraza (10), saglasno oznakama (7), neposredno dobijamo rekurentne obrasce za stepenovanje neprigušene dinamičke matrice:

$$\left|\begin{array}{l} C_m = K_{m-1} \\ K_m = C_{m-1}K \\ I_m = C_{m-1} \\ O_m = K_{m-1} \end{array}\right. \qquad (m=2,3,\ldots,L) \qquad (12)$$

BROJNI PRIMER

Za sistem sa jednim stepenom slobode (sl.2), uzet prema /2/, odredimo R i R^3.



$$R = \left[\begin{array}{c|c} -m^{-1}c & -m^{-1}k \\ \hline 1 & 0 \end{array}\right]$$

SL.2 SISTEM I NJEGOVA MATRICA

U slučaju prigušenih vibracija (c=0.5), saglasno izrazima (6) i (10), matrice R i R^3 glase:

$$R = \left[\begin{array}{c|c} -0.5 & -1. \\ \hline 1. & 0. \end{array}\right]; \quad R^3 = \left[\begin{array}{c|c} 0.875 & 0.75 \\ \hline -0.75 & 0.5 \end{array}\right].$$

S druge strane, u slučaju neprigušenih vibracija (c=0), saglasno izrazima (11) i (12), imamo:

$$R = \left[\begin{array}{c|c} 0. & -1. \\ \hline 1. & 0. \end{array}\right]; \quad R^3 = \left[\begin{array}{c|c} 0. & 1. \\ \hline -1. & 0. \end{array}\right].$$

ZAKLJUČAK

Kao specijalan slučaj matričnog stepenovanja preko četiri kvadratne podmatrice, izvedeni su rekurentni obrasci za stepenovanje kako prigušene, tako i neprigušene dinamičke matrice diskretnih sistema.

Pogodnost izvedenih obrazaca je što se, zbog posebne strukture dinamičke matrice, primenom podmatrica smanjuje broj skalarnih pri stepenovanju.

LITERATURA

/1/ Furundžić, S., (1981): "Jedan numerički postupak dinamike konstrukcija", XV jug. kongr. mehanike, Kupari.

/2/ Furundžić, S., (1981): "Jedan algoritam i program za računanje dinamičkog odgovora konstrukcija", Simpozijum Informatica 81, Ljubljana.

POSLOVNI SISTEMI

BUSINESS SYSTEMS

PROJEKT LOGIČKE STRUKTURE BAZE PODATAKA O BIBLIOTECI

DALIBOR MILENKOVIC
IBK - VINČA INSTITUT ZA INFORMATIKU
"Mihailo Petrović Alas"
BEOGRAD JUGOSLAVIJA

UDK: 681.3.01

ABSTRAKT. Na osnovu opisa funkcionisanja biblioteke tehnikom strukturne sistem analize konceptualizovan je sistem u formi dijagrama toka podataka. Identifikovani su objekti, relacije i atributi sistema i primenjen je postupak normalizacije za generisanje logičke strukture baze podataka o biblioteci sa UDK metodom klasifikacije. Zbog ograničenja UDK metode analizirana je struktura tezaurusa i istom metodologijom kreiran je datološki model tezaurusa koji je zatim integrisan u celoviti datološki model biblioteke.

DESIGN OF LIBRARY DATA-BASE LOGICAL STRUCTURE. Based on the description of library functioning by means of a structured system analysis a library system is conceptualized in the form of data flow charts. The system objects, relations and attributes are identified and normalizing procedure applied for generation of data-base logical structure of the library with UDC classification method. Due to limitations of UDC method thesaurus structure is analized and the same methodology applied for a thesaurus data model creation, which is thereafter integrated into the overall library data model.

1. Uvod

Iako je krajnji cilj projektovanja informacionog sistema (IS) realizacija na računaru, u fazi logičkog projektovanja baze podataka (BP), kao centralne komponente informacionog sistema, projektant mora biti daleko od pomisli na računar i fizičku implementaciju. Ovo načelo obezbedjuje potrebnu nezavisnost rešenja u odnosu na konkretni softver za upravljanje bazom podataka (SUBP). Metodologija projektovanja IS bazira se na sistemskom pristupu rešavanja problema koji je zasnovan na opštoj teoriji sistema i uključuje faze:

- verbalnog opisa problema;
- konceptualizacije grafičkim metodama i
- formulacije problema u okviru opšte teorije sistema.

U ovom radu disciplinovano je provedena ova metodologija kroz razvoj logičke strukture BP o biblioteci koja uz konvencionalnu UDK metodu klasifikacije integriše i metodu koordinantnog indeksiranja publikacija preko modela tezaurusa. Za izgradnju kompletnog IS potrebno je još obaviti sledeće aktivnosti:

- konvertovati i realizovoati logičku strukturu BP na nekom konkretnom SUBP kao što su I-D-S/II, TOTAL, IMS i
- projektovati programe za ažuriranje i uvid (izveštavanje) u paketnoj i transakcionoj obradi.

2. Opis bibliotečkog sistema

Sve biblioteke, od onih opšte namene do INDOK--biblioteka, imaju istu funkciju i strukturu. Osnovne bibliotekarske delatnosti uključuju nabavku bibliotečke gradje, katalogizaciju, organizaciju bibliotečkih fondova i službi za čitaoce. Ove delatnosti odvijaju se na dva jasno odvojena toka: prvi se sastoji od novih knjiga i periodike koje biblioteka prima i naziva se "put nove knjige", dok se drugi naziva "put po želji citalaca" i sastoji od literature koja se izdaje na zahtev čitalaca [1].

Put nove knjige počinje sa prispećem knjige odn. primarnog dokumenta u biblioteku i njenom analitičko - sintetičkom obradom. Rezultat obrade je sekundarni dokument - kataloška kartica gde se knjiga jedinstveno identifikuje tzv. signaturom. Slede ostali bibliografski podaci, kao i podaci o klasiranju u oblast kojoj knjiga pripada prema usvojenom metodu klasifikacije npr. univerzalna decimalna klasifikacija (UDK). Signatura se unosi i na knjigu koja se zatim arhivira na police dok se kataloška kartica umnožava u potrebnom broju primeraka i ulaže u kataloge: autorski, naslovni i predmetno-stručni. Struktura ovog poslednjeg zasniva se na metodu klasifikacije koji je biblioteka prihvatila. Periodično bibliotekari izradjuju posebnu vrstu sekundarnih dokumenata - bibliografije novoprispelih knjiga po navedenim katalozima. Time se put nove knjige završava.

Sa arhiviranjem knjige počinje njen drugi put, put po zahtevu čitalaca. Iz jednog od kataloga čitalac nalazi signature knjiga koje ga interesuju i ako su slobodne dobija ih na čitanje. Odsek za pozajmicu vodi kartoteku čitalaca koja sadrži podatke o izdatim knjigama, te kartoteku s kartonima knjiga na kojima su zapisana imena ili šifre citalaca kojima je izdata odredjena knjiga. Ako je knjiga zauzeta može se izvršiti rezervacija. Po povratku, pozajmljene knjige se, ako nisu rezervisane, vraćaju na police, a ako jesu, čitalac koji ih je rezervisao se obaveštava da ih moze podići. Ovim se završava druga stalna cirkulacija bibliotečke gradje.

3. Strukturna sistem analiza

Na osnovu verbalnog opisa bibliotečkog sistema tehnikom strukturne sistem analize [2] konceptualizovan je sistem u formi dijagrama toka podataka (Sl.1) Time je dobijen veran model, analiziranog sistema jer uključuje u sebi izvore, "skladišta" i odredišta podataka, tokove podataka i procese koji ih transformišu, a ipak ostaje na nivou logike sistema ne upuštajuci se u fizičku implementaciju. Najzad, uz tok podataka uključen je i materijalni tok - put knjige, budući da je dobar deo tokova podataka posledica cirkulacije knjige.

Numeracija procesa na dijagramu (Sl.1) ne izražava njihovu striktnu sukcesivnost. Napred izložena dva puta knjige obuhvataju sledeće procese:

- put nove knjige: 1 i 2;
- put po zahtevu čitalaca: 4, 5, 6, 7 i 8.

Analizirajući strukture podataka na tokovima u odnosu na datoteke i procese kao mesta spremanja i generisanja podataka, te definisanjem sadržaja datoteka uspostavlja se rečnik podataka posmatranog sistema kao krajnji cilj sistem analize.

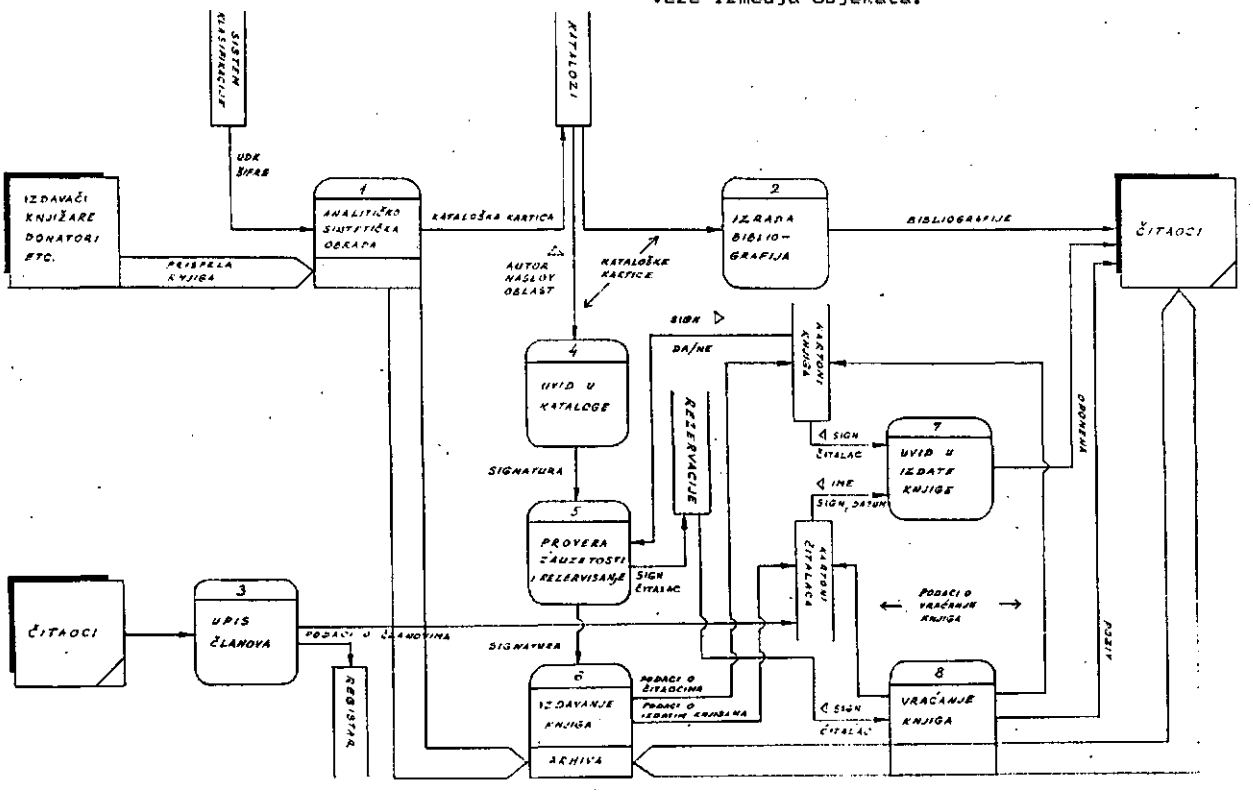
4. Logička struktura BP o biblioteci sa UDK metodom klasifikacije

U sledećem koraku sistemskog pristupa - modeliranju uvodimo pojam datološkog modela sistema kao skupa podataka i njihovih medjusobnih veza pomoću koga je moguće preneti ili sačuvati informacije o datom sistemu. Poznavanjem objekata i njihovih medjuzavisnosti u bibliotečkom sistemu i njihovim opisivanjem pomoću skupa atributa direktno se modelira konceptualni datološki model posmatranog sistema. Ovako dobijeni model nezavisan je od procesa obrade podataka koji će se nad njim vršiti. Polazi se od pretpostavke da je datološki model veran odraz sistema i da će kao takav moći da pruži sve informacije koje se mogu zahtevati u realnom sistemu.

Prethodne dve faze omogućavaju da se, kroz izgradnju rečnika podataka, identifikuju ovi objekti i atributi bibliotečkog sistema:

OBJEKTI	ATRIBUTI
OBLAST	oznaka oblasti (UDK), naziv (NO)
KNJIGA	signatura (SIG), autor (A), naslov (NA), impresum (IM), datum prispeća (DP)
ČITALAC	broj čitaoca (BC), ime (IME), br. lične karte (BLK), adresa (AD), zanimanje (ZAN), članarina (CL)
REZERVACIJA	signatura (SIG), rezervisao (BC1)

Veze izmedju objekata:



Slika 1 - Dijagram toka podataka (i cirkulacije knjiga) u biblioteci

ČITALAC - KNJIGA: čitalac se zadužuje sa 1-3 knjige. Atribut ove veze je datum uzimanja knjige (DU).
OBLAST - KNJIGA: knjige su svrstane po oblastima po metodi UDK.

Struktura oblasti po metodi UDK [1] je hijerarhijska pa nam rekurzivna definicija stabla omogućava da se ova homogena struktura predstavi preslikavanjem M:1 izmedju oblasti i fiktivnog objekta PODOBLAST koji je jedinstveno identifikovan atributom - oznaka podoblasti (UDK1).

Izmedju elementarnih podataka, označenih skraćenicama, koji na nivou datološkog modela odgovaraju atributima objekata i relacija, postoje sledeća relevantna preslikavanja:

1.	((UDK), (NO))	(1:1)
2.	((SIG), (A))	(1:M)
3.	((SIG), (NA))	(1:M)
4.	((SIG), (IM))	(1:M)
5.	((SIG), (DP))	(1:M)
6.	((BC), (IME))	(1:M)
7.	((BC), (BLK))	(1:1)
8.	((BC), (AD))	(1:M)
9.	((BC), (ZAN))	(1:M)
10.	((BC), (CL))	(C:M)
11.	((SIG), (BC1))	(M:M)
12.	((BC), (SIG))	(M:1)
13.	((SIG), (DU))	(1:M)
14.	((UDK), (SIG))	(M:1)
15.	((UDK), (UDK1))	(M:1)

Na osnovu 1. pravila za struktuiranje logičkih zapisa [3], da se izmedju ključa i opisnih podataka u okviru jednog logičkog zapisa mogu realizovati preslikavanja tipa (1:1), (1:M), (C:1) i (C:M), dolazi se do sledećeg skupa logičkih zapisa:

OBLAST (UDK, NO)
PODOBLAST (UDK1)
KNJIGA (SIG, A, NA, IM, DP, DU)
ČITALAC (BC, IME, BLK, AD, ZAN, ČL)

Pravila 2 i 3 o nepotpunoj funkcionalnoj i tranzitivnoj zavisnosti opisnog podatka od ključa su zadovoljena pa se zapisi dalje ne razbijaju. Ovi zapisi ne obuhvataju preslikavanja 11, 12, 14 i 15 koja se na osnovu 5. pravila realizuju izmedju zapisa. Osim toga, preslikavanje tipa (M:M) transformiše se u dva preslikavanja tipa (M:1) uvodjenjem novog logičkog zapisa čiji je ključ složeni ključ sastavljen od ključeva zapisa koji su bili vezani preslikavanjem (M:M):

11.a ((SIG), (SIG,BC1)) (M:1)
11.b ((BC1), (SIG,BC1)) (M:1)

5. Zahtevi koji se postavljaju pred BP

Rezultirajući konceptualni datološki model odn. logička struktura BP o biblioteci prikazana je na dijagramu sa izdućenim blokovima [4] (S1.2). Veze izmedju zapisa kao i sekundarni ključevi predstavljeni su linijama na kojima strelice označavaju tip asocijacije:
> asocijacija tipa 1 >> asocijacija tipa M

Baza podataka treba da pruži, kako to pokazuje dijagram toka podataka odgovor na niz pitanja. Analizirajmo na nekoliko karakterističnih zahteva kako se dolazi do odgovora sa stanovišta kretanja po prikazanoj mrežnoj strukturi ucrtanim putanjama i pristupa logičkim zapisima i pojedinim elementarnim poljima.

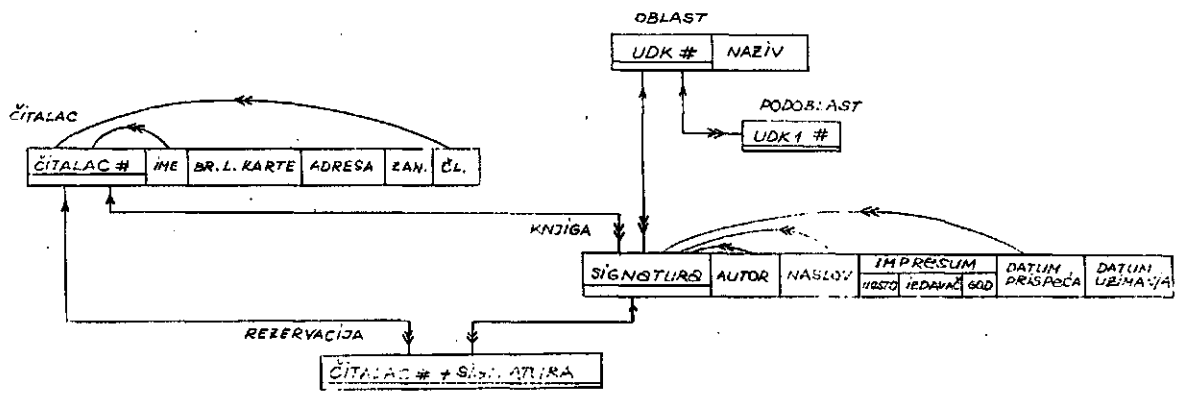
Uvid u stručni katalog ostvaruje se po sistemu menu-ja jer je struktura ovog kataloga idealna za ovakav prikaz, a osim toga od čitaoca se ne može ocekivati da poznaje plan UDK oznaka. Prolaz kroz stablo oblasti ostvaruje se u dva repetitivna koraka. Na osnovu datog UDK broja oblasti pristupa se zapisu PODOBLAST i pronalaze svi UDK1 brojevi koji su za njega vezani, a zatim se izabranoj podoblasti pristupa tako što se njen UDK1 broj sada tretira kao UDK broj oblasti.

Za ilustraciju namene sekundarnih ključeva neka se pretpostavi da korisnik IS biblioteke želi uvid u autorski katalog sa ciljem da dobije signaturu knjige čijeg autora zna. Zahvaljujući sekundarnom ključu AUTOR biće pronadjen ne jedan jedinstven zapis, već svi zapisi koji imaju odredjeno svojstvo: u ovom primeru zapisi knjiga jednog autora.

Veza izmedju zapisa ČITALAC i KNJIGA omogućava bibliotekarima da uvek znaju koje je knjige pozajmio neki čitalac i kada ih mora vratiti odn. ko čita knjigu koja nije na polici, a koju je zatražio neki drugi čitalac. U drugom slučaju pristupa se zapisu KNJIGA i kad se utvrdi da je polje DATUM UZIMANJA popunjeno ide se do sloga ČITALAC i utvrdjuje kod koga se nalazi tražena knjiga.

6. Metoda koordinantnog indeksiranja

Sistem klasifikacije po UDK metodi modeliran je imajući u vidu da je to široko rasprostranjeni, a osim toga i obavezni sistem klasifikacije. Medjutim, klasifikacija dokumenata po samo jednoj oznaci predstavlja ozbiljno ograničenje, posebno kada je reč o naučnim i tehničkim dokumentima.



Slika 2 - Logička struktura BP o biblioteci

Prevazilaženje ovog nedostatka pruža metoda koordinantnog indeksiranja sa idejom da se sadržaj svakog dokumenta opiše jednim skupom ključnih reči. Ključne reči mogu se shvatiti kao koordinate jednog višedimenzionog prostora u kome svaka tačka predstavlja neki koncept ili ideju date oblasti. Otuda se za svaki dokument mogu odrediti te njegove koordinate tj. ključne reči i obrnuto, ako se podje od ključnih reči tj. koordinata može se doći do tačke u prostoru tj. dokumenta koji sadrži traženi koncept.

Osnovu ove metode čini tezaurus - terminološki rečnik odredjene naučne oblasti pri čijem je standardizovanju potrebno zadovoljiti uslove koji se odnose na:

a) preciznost i jednoznačnost ključnih reči;
b) kontrolisanu upotrebu sinonima;
c) eliminisanje polisemije i homonima i
d) uspostavljanje paradigmatskih (pojmovnih) odnosa izmedju ključnih reči.

Skup deskriptora (standardizovanih ključnih reči) za datu oblast ima vrlo složenu mrežnu organizaciju koja ima karakter zakonitosti za skupove semantičkih elemenata. Mada postoji niz unutrašnjih odnosa u ovakvom pulu deskriptora, za korisnika je prvenstveno važno da deskriptore vidi grupisane po srodnosti svog značenja. S tim ciljem iz pula se zahvata niz deskriptora koji čine facetu, a facete su organizovane po semantičkim poljima kao kategorijalnim pojmovima date oblasti. Odavde proizlazi da delove tezaurusa sačinjavaju:

- spisak semantičkih polja i njihovih faceta;
- spisak deskriptora po facetama i
- alfabetski spisak deskriptora sa njihovim indikatorima.

7. Indikatori medjusobnih odnosa deskriptora

Struktura tezaurusa počiva na dve vrste ovih indikatora koji obezbedjuju napred pomenute uslove a) - c) i d) respektivno. U prvu grupu spadaju sledeći indikatori:

USE (upotrebi). Kod izgradnje tezaurusa, iz popisa ključnih reči izdvajaju se grupe ključnih reči koje se smatraju sinonimima. Iz svake grupe uzima se jedna reč za reprezentaciju cele grupe i ona postaje deskriptor. Sve ostale reči u grupi proglašavaju se nedeskriptorima i povezuju se sa predstavnikom grupe indikatorom USE a ovaj sa njima indikatorom:

UF (used for = upotrebljen za).

SN (scope note = napomena o opsegu/definicija.) Uopšteno gledajuci ovaj se indikator daje da bi se izbegla dvosmislenost. Ali ima i značajniju ulogu: da označi opseg jednog pojma. Ove definicije nisu definicije rečnika već kratki opisi smisla koji se daje jednom pojmu.

Indikatori druge grupe uspostavljaju semantičku mrežu u skupu deskriptora:

BT (broader term = širi pojam). Indikator BT ukazuje na opštiji deskriptor koji je nadredjen datom, i to samo jedan (ne razmatra se polihijerarhija), čime uspostavlja generički odnos.

NT (narrower term = uži pojam). Nasuprot indikatoru BT ovaj indikator ukazuje na specifičnije deskriptore koji su podredjeni datom i uspostavlja odnos vrste. Na taj način par indikatora BT-NT definiše isečak hijerarhijskog lanca za posmatrani deskriptor.

RT (related term = srodni pojam). Pored opisanih hijerarhijskih u najvažnije paradigmatske odnose spadaju: - funkcionalna sličnost;
- uzročno-posledični odnosi i
- odnosi deo-celina.

Indikatorom RT obuhvaćeni su ovi odnosi čime je korisniku tezaurusa omogućen bolji izbor deskriptora.

Uz svaki deskriptor ide i njegov frekventni broj koji registruje u koliko je dokumenata deskriptor upotrebljen za indeksiranje i koristi se za predvidjanje odziva sistema. Primer jednog deskriptora i njemu pridruženih pojmova [6]:

DECONTAMINATION [3576; 3576]
UF *radioactive decontamination*
BT1 cleaning
RT bioadsorbents
RT radiation protection
RT safety showers

U domen ovog poglavlja spadaju i tzv. zajednički (link) indikatori. To su brojevi koji povezuju one deskriptore čija koordinacija predstavlja predmet o kome je reč u dokumentu i sledstveno sprečavaju pretraživanje bazirano na lažnoj koordinaciji deskriptora pridruženih dokumentu.

8. Konceptualni datoloski model tezaurusa

Na osnovu analizirane strukture mogu se definisati ovi objekti i atributi tezaurusa:

OBJEKTI	ATRIBUTI
SEMANTIČKO POLJE	šifra semantičkog polja (SSP) naziv s.p. (SMNT)
FACETA	šifra facete (SF), naziv (FAC)
DESKRIPTOR	šifra deskriptora (REC), deskriptor (D), scope note (SN), frekventni broj (FB)
NEDESKRIPTOR	šifra sinonima (SIN), sinonim (ND)
ŠIRI/UŽI POJAM	šifra pojma (BNT)
SRODNI POJAM	šifra pojma (RT)
KNJIGA	signatura (SIG), ..., abstrakt (ABS)

U praksi je sa indeksiranjem dokumenata povezano i postojanje abstrakta tih dokumenata pa otuda proširenje objekta KNJIGA odn. ovde bi već bilo preciznije DOKUMENT. Važno je napomenuti da je zbog različitih vrsta primarnih dokumenata slog bibliografskog opisa realno složeniji, a pomeranjem fokusa prema bibliografskim servisima može biti još složeniji pa i predstavljen posebnim slogovima.

Veze izmedju objekata su brojne:
SEMANTIČKO POLJE - FACETA: semantičko polje sadrži niz faceta.
FACETA - DESKRIPTOR: faceta sadrži max 15 deskriptora.
DESKRIPTOR - NEDESKRIPTOR: deskriptor može imati više sinonima.
Naziv ove veze u smeru NEDESKRIPTORA daje indikator UF, a u obrnutom smeru indikator USE.
DOKUMENT - DESKRIPTOR: dokumenti su indeksirani sa 5-15 deskriptora, po pravilu. Jedan deskriptor se javlja kao indeks većeg broja dokumenata. Atribut ove veze su zajednički indikatori (ZI).

U pulu deskriptora indikatori BT-NT i RT formiraju dve homogene strukture. Prva je hijerarhijska i definiše se vezom DESKRIPTOR - ŠIRI/UŽI POJAM, a druga je mrežna i definiše se vezom DESKRIPTOR - SRODNI POJAM.

Relevantna preslikavanja su:

1. ((SSP), (SMNT))	(1:1)
2. ((SF), (FAC))	(1:1)
3. ((REC), (D))	(1:1)
4. ((REC), (SN))	(C:1)
5. ((REC), (FB))	(1:M)
6. ((SIN), (ND))	(1:1)
7. ((SIG), (ABS)	(C:1)
8. ((SSP), (SF))	(M:1)
9. ((SF), (REC))	(M:1)
10. ((REC), (SIN))	(M:1)
11. ((REC), (BNT))	(M:1)
12. ((REC), (RT))	(M:M)
13. ((SIG), (REC))	(M:M)
14. ((SIG,REC), (ZI))	(M:1)

Postupkom normalizacije dobija se sledeći skup logičkih zapisa:

SEMANTIČKO POLJE (SSP, SMNT)
FACETA (SF, FAC)
DESKRIPTOR (REC, D, SN, FB)
NEDESKRIPTOR (SIN, ND)
ŠIRI/UŽI POJAM (BNT)
SRODNI POJAM (REC+RT)
VEZA (SIG+REC)
ZAJEDNIČKI INDIKATORI (ZI)
DOKUMENT (SIG, ..., ABS)

Ovaj skup prikazan je dijagramom sa izduženim blokovima (Sl.3) na kome su strelicama eksplicitno označeni elementarni podaci DESKRIPTOR i SINONIM kao sekundarni ključevi da bi se preko naziva (ne)deskriptora omogućio pristup u BP.

9. Zaključna razmatranja

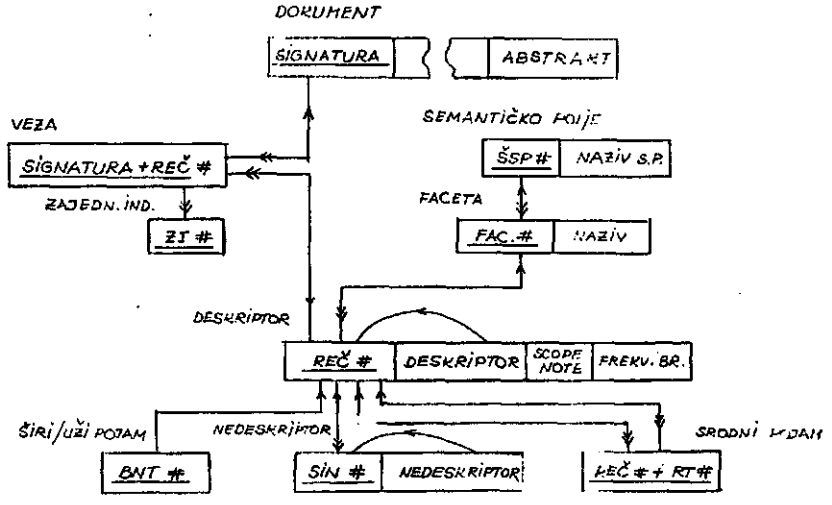
Kreirani model tezaurusa podržava ključne funkcije koordinantnog indeksiranja:

- Formiranje, štampanje i ažuriranje tezaurusa.
- Indeksiranje primarnog dokumenta uključiv i automatsko dodeljivanje dokumentu svih širih pojmova svakog izabranog deskriptora.
- Pretraživanje BP da bi se pružio odgovor na korisnikov zahtev za publikacijama. Zahtev za pretraživanje - profil postavlja se u vidu logičke kombinacije deskriptora. U sklopu programa za pretraživanje vrši se predvidjanje odziva sistema korišćenjem frekventnog broja.
- Prikaz/štampa sekundarnog dokumenta (bibliografskog opisa).

Datološki model tezaurusa uklapa se u datološki model biblioteke preko pomoćnog zapisa koji povezuje zapise DESKRIPTOR i DOKUMENT. Ovo integrisanje ne zahteva bitnije izmene IS biblioteke pa tezaurus koji pokriva odredjene naučne oblasti može postojati uporedo sa konvencionalnim sistemom klasifikacije.

10. Literatura

[1] A.I. Mihailov, R.S. Giljarevskij - UVOD U INFORMATIKU/DOKUMENTACIJU, Zagreb, Referalni centar sveučilišta, 1977.
[2] C. Gane, F. Sarson - STRUCTURED SYSTEM ANALYSIS, Englewood Cliffs, N.J., Prentice-Hall, 1979.
[3] B. Lazarević V. Jovanović - JEDAN PRISTUP PROJEKTOVANJU INFORMACIONOG SISTEMA, Praksa, 1981, god. XVI, br. 5
[4] J. Martin - COMPUTER DATA-BASE ORGANIZATION, (2. ed.), Englewood Cliffs, N.J., Prentice-Hall, 1977.
[5] C. Todeschini - INIS: MANUAL FOR INDEXING, Vienna, IAEA, 1974.
[6] IAEA - INIS: THESAURUS, Vienna, IAEA, 1985.
[7] - THESAURUS DE L'EDUCATION UNESCO: BIE, Paris, Les Presses de l'Unesco, 1976.



Slika 3 - Konceptualni datološki model tezaurusa

PODATKOVNE ZBIRKE

Tatjana WELZER
TEHNIŠKA FAKULTETA MARIBOR

UDK: 681.3.016

POVZETEK - Referat opisuje različne strukture podatkovnih zbirk in njihovo oblikovanje. Opisana je drevesna, mrežna in relacijska struktura zbirke ter metoda za njeno oblikovanje.

DATA BASES: This paper describes different structures of data bases and data bases design. Tree, network and relation data bases and a method for data base design are described.

1. UVOD

Širjenje distribuiranih računalniških sistemov in njihovo povezovanje v lokalne in javne računalniške mreže, omogoča novo in boljšo organiziranost splošnih informacijskih sistemov. Zgradba uspešnih računalniško podprtih sistemov temelji na organiziranih podatkovnih zbirkah. Podatkovne zbirke so lahko centralizirane ali distribuirane. V centralizirani podatkovni zbirki se vsi podatki (trenutne vrednosti) nahajajo na eni lokaciji. Distribuirana podatkovna zbirka pa predvideva podatkovno zbirko, ki se tekoče obnavlja. Razdeljena je na več delov z ločenim, vendar enakovrednim upravljanjem. Uporabnik se bo avtomatsko povezal s tisto enoto, kjer so zahtevani podatki.

2. PODATKOVNE ZBIRKE

Podatkovno zbirko definiramo kot neredundantno zbirko vzajemno povezanih podatkov, ki jih uporabljamo za izvajanje ene ali več aplikacij ali pa kot zbirko podatkov, ki so medsebojno povezani brez nepotrebne redundance in omogočajo optimalno izvajanje najrazličnejših aplikacij. Podatkovno zbirko lahko definiramo še na več načinov, bistvo vseh definicij pa je, da nam podatkovna zbirka predstavlja skladišče informacij, ki jih uporabljamo za izvajanje posameznih aplikacij.

V razvoju podatkovnih zbirk od njihovega nastanka do danes so se izoblikovale tri osnovne strukture: hierarhična ali drevesna, mrežna in relacijska struktura.

2.1. DREVESNA STRUKTURA

Drevo je zgrajeno hierarhično iz elementov imenovanih segmenti, med katerimi veljajo naslednji odnosi:

1. Najvišji nivo v hirerarhiji ima le en segment, ki ga imenujemo koren.
2. Vsi segmenti razen korena so povezani z enim in to samo enim segmentom na višjem nivoju, gledano s stališča danega nivoja.

Segment na višjem nivoju imenujemo starši, segmenti na danem nivoju so otroci. Posamezen element lahko istočasno opravlja nalogo staršev in otrok.

Drevesna struktura je lahko uravnotežena ali neuravnotežena. Pri uravnoteženem drevesu ima vsak segment enako število vej. Razvoj drevesa pri tem poteka od vrha proti dnu in od leve proti desni. Poseben primer uravnoteženega drevesa je binarno drevo, pri katerem iz vsakega segmenta izhajata natanko dve veji ali nobena veja.

Ob binarnem drevesu poznamo še B-drevo. B-drevo je uravnoteženo, vendar organizirano v vejah končne dolžine, ki lahko vsebujejo le določeno število informacij. Ko vejo napolnimo z elementi, se preprosto razcepi v dve novi veji na istem nivoju.

2.2. STRUKTURA MREŽE

Če ima otrok več staršev in povezave ne moremo opisati s pomočjo drevesa, govorimo o mrežni strukturi. V njej je lahko vsak segment povezan z vsakim segmentom. Vsako mrežno strukturo je možno poenostaviti, tako da jo prevedemo v preprostejšo drevesno strukturo. To poenostavitev izvedemo s pomočjo redundance(v zapletenejših primerih se zato poenostavitvi odpovemo).

2.3. STRUKTURA RELACIJE

Pri klasični strukturi podatkovne zbirke (drevesna in mrežna struktura), najpogosteje naletimo na naslednje probleme: podatkovne strukture na logičnem nivoju so zapletene, ni podatkovne neodvisnosti v smeri fizično - - logično, manjka preprost mehanizem za dostop do podatkov na logičnem nivoju, programer pa je prisiljen uporabljati vnaprej definirane fizične poti. Rešitev teh problemov predstavlja relacijska podatkovna zbirka. V njej so vsi podatki eksplicitno predstavljeni kot vrednosti v tabelah-relacijah, ki so osnova relacijske strukture. S tabelami zelo enostavno prikažemo relacije - povezave med posameznimi podatki.

PREDMETI	PŠTEV	PIME	PLET	PŠTEVILO
	P1	OSNOVE ELEKTR.	1	90
	P2	MATEMATIKA	3	60
	P3	ELEKTRONIKA	2	75
	P4	REGULACIJE	2	75
	P5	FIZIKA	1	45

UČITELJI	UŠTEV	UIME	NAZIV	KRAJ
	Oš1	NOVAK	DOC	MARIBOR
	Oš2	PETAN	R. PROF	LJUBLJANA
	Oš3	KOREN	V. PRED	KRANJ

RAZPOREDITEV	UŠTEV	PŠTEV	DATUM	ŠT. TEDNOV
	Oš1	P4	05.10.83	10
	Oš2	P1	18.02.84	15
	Oš3	P2	15.11.83	10
	Oš3	P5	05.10.83	05

Vsaka tabela ima svoje ime. V našem primeru so to PREDMETI, UČITELJI in RAZPOREDITEV. Vsaka vrstica nam opisuje eno skupino podatkov. Posamezne lastnosti so opisane v stolpcih, katerih imena imenujemo atribute. Množico vseh vrednosti, ki nastopi v posameznem stolpcu pa imenujemo domena. V tabeli PREDMETI so atributi PŠTEV, PIME, PLET in PŠTEVILO UR. Ti atributi opisujejo za vsak predmet: njegovo številko oziroma oznako, ime, letnik v katerem se predmet predava in število ur. Če želimo ime stolpca ločiti od morebitnega istega imena v kakšni drugi tabeli, mu kot predpono dodamo ime tabele s piko (PREDMETI.PLET).

Podatku, ki nedvoumno določa vrstico v tabeli pravimo ključ. Lahko je enostaven, če je v enem stolpcu, oziroma sestavljen, če ga določa več stolpcev. Ključa PŠTEV v tabeli PREDMETI in UŠTEV v tabeli UČITELJI sta enostavna. V tabeli RAZPOREDITEV pa je ključ lahko sestavljen iz UŠTEV, PŠTEV in DATUM.

Iz opisanih primerov lahko spoznamo tudi bistvene lastnosti relacij:

V posamezni tabeli nimamo podvojenih vrstic, niti podvojenih stolpcev. Vrstni red vrstic in stolpcev ni pomemben. Med različnimi tabelami ni vidnih vezi, saj povezave med njimi dosežemo s primerjanjem vrednosti v posameznih stolpcih.

2.3.1. OPERACIJE V RELACIJSKIH MODELIH

Osnovne relacije so podane v tabelah, iz katerih lahko s primernimi relacijskimi operatorji gradimo nove relacije - tabele (relacija in tabela sta sinonima).

Pravila delovanja relacijskih operatorjev nam pojasnjuje relacijska algebra. Operatorji (unija, presek, razlika in kartezični produkt) so sposojeni iz teorije množic. Ob njih v relacijski algebri srečamo še druge operatorje med katerimi so najpomembnejši selekcija, projekcija in združitev.

Selekcija

S tem operatorjem dobimo iz tabele A tabelo B tako, da iz prve tabele prenesemo v drugo tabelo samo tiste vrstice, ki ustrezajo nekemu pogoju.

Projekcija

Operator projekcije prenese iz osnovne tabele samo določene stolpce. Pri tem se lahko zgodi, da dobimo v novi tabeli dve ali več enakih vrstic. Redundančne vrstice izločimo in s tem izpolnimo zahtevani pogoj za relacijsko strukturo.

Združitev

Pod tem imenom srečamo več različnih operatorjev: združitev po enakosti, naravno združitev in zunanjo naravno združitev.

Imejmo relaciji R in S, ki imata skupno domeno D. Z A in B označimo atributa v relacijah R in S in oba definiramo nad D. Združitev po enakosti relacije R na A z relacijo S na B je takšna podmnožica kartezičnega produkta RxS, kjer ima vsaka vrstica iste vrednosti v stolpcih A in B.

O naravni združitvi govorimo, kadar obdržimo samo en stolpec. Ta se zgodi, če imamo po združitvi po enakosti dva stolpca z istimi vrednostmi. Zunanja naravna združitev pa nam omogoča, da v relacijo uvrstimo tudi tiste vrstice iz združenih tabel, ki nimajo ustreznega para v drugi tabeli. Opisane operatorje lahko med seboj tudi povežemo in dobimo povezane operatorje.

3. OBLIKOVANJE PODATKOVNIH ZBIRK

Z razvojem podatkovnih zbirk se je močno spremenilo tudi njihovo oblikovanje. Sprva je bil precejšen del oblikovanja povezan s fizično lokacijo pomnenja podatkov, z njihovim nalaganjem in dostopom do sekundarnega pomnilnika. Te naloge je danes v glavnem prevzel sistem za upravljanje podatkovne zbirke, tendenca oblikovanja podatkovnih zbirk pa je v optimizaciji logičnih modelov podatkovnih zbirk.

Pri oblikovanju podatkovne zbirke najprej analiziramo ogrodje (obstoječ sistem), nato pa postopoma prehajamo do posameznih nalog, podnalog in podatkovnih elementov, ki nas zanimajo. Nalogo opravimo postopoma v štirih fazah:

1. V analizi okolja zahtev zbiramo informacije, ki so potrebne za izvajanje naslednjih korakov. Osnovne tehnike za pridobivanje potrebnih informacij so prebiranje dokumentov in drugih pisanih virov, izpraševanje ljudi in analiziranje anket. To je istočasno tudi vhod v prvo fazo, njen izhod pa predstavlja uporabna specifikacija v obliki diagrama poteka, ki nam predstavi najpomembnejše aktivnosti sistema in njihove medsebojne povezave.

2. V drugi fazi analiziramo in specificiramo sistem. Izhajamo iz obstoječega diagrama, ki nam zagotavlja dobro poznavanje ciljnega sistema. Vsako aktivnost diagrama hierarhično razdelimo v več posameznih in delnih nalog. Pri tem procesu pride tudi do delitve podatkov, ki pripadajo posamezni aktivnosti. Razdelimo jih v podatkovne elemente ali pa v podmnožice le teh. Delitev aktivnosti in podatkov poteka tako dolgo, da imámo samo še enostavne in razumljive naloge ter podatkovne elemente. Končni rezultat je diagram poteka, ki je tokrat podrobneje razdelan.

   Temu prvemu delu faze sledi še specifikacija nalog, podatkov (dokumentov) in uporabniških programov.

3. Cilj zasnove modela je prenos znanja, ki smo ga zbrali v 1. in 2. fazi, v določenen prikaz. Za uspešno modeliranje uporabljamo različne jezike (CSDL, Taxis, D-graphs, E-R model).

4. V zadnji fazi oblikujemo logično shemo. Izvesti moramo naslednje naloge:

   Izdelamo natančni model podatkovne zbirke na osnovi koncepta modela, generiramo bazično logično shemo,

integriramo novo shemo, ki predstavlja model povezav in njihovo komunikacijsko težo, optimiziramo shemo in realiziramo logično shemo ob pomoči optimizacijske sheme.

Ko uspešno zaključimo zadnjo nalogo četrte faze, je podatkovna zbirka pripravljena za izvajanje operacijske faze.

4. ZAKLJUČEK

Pregled razvoja, struktur in oblikovanja podatkovnih zbirk je pokazal, da so na tem področju spremembe pogoste.

Tako so na področju podatkovnih zbirk trenutno izredno "popularne" relacijske zbirke, ki tudi nepoznavalcem organizacije podatkovne zbirke omogočajo aktivno vključevanje v oblikovanje le te.

Kot pomoč pri oblikovanju in delu z relacijskimi podatkovnimi zbirkami pa je nadvse pomembna tudi umetna inteligenca in njeni jeziki (LISP, PROLOG, POP, micro LISP, micro PROLOG...). Prolog lahko direktno uporabimo kot interpreter za relacijski model podatkovne zbirke. Še boljše pogoje za dela pa daje POPLOG- programirano okolje v katerem so združeni LISP, PROLOG in POP.

5. LITERATURA

1. S. Algić: Relacione baze podataka, Sarajevo 1984

2. D. N. Chorafas: Database for networks and minicomputers, New York 1982

3. J. Martin: Computer Data-Base Organization, New Yersey 1975

4. G. Wiederhold: Database Design, Tokyo 1981

5. G. Wiederhold: Knowledge and Database Management, IEEE Software januar 1984

O SINTEZI GENERALIZIRANEGA VREDNOTENJA INFORMACIJSKIH SISTEMOV

Viljem RUPNIK
EKONOMSKA FAKULTETA BORISA KIDRIČA
Ljubljana, Kardeljeva ploščad 17
Yugoslavia

UDK: 681.3.007

P O V Z E T E K

Referat obravnava multidimenzionalno vrednotenje informacijskih sistemov v funkcionalno strukturirani obliki. Pristop omogoča povezavo z običajnim knjigovodstvom in dovoljuje specifične algebre za posamezne komponente vrednotenja.

ON THE SYNTHESIS OF GENERALISED EVALUATION OF INFORMATION SYSTEMS

A paper deals with multidimensional evaluation of information systems in a functionally structured form. The approach enables us to couple with ordinary book keeping and allows specific algebras for particular components of evaluation.

Večdimenzionalno vrednotenje informacijskih sistemov nas kaj hitro pripelje do generaliziranih stroškovnih in vrednostnih komponent pri procesu vrednotenja (evoluacije)/1/Naloga generaliziranega vrednotenja informacijskih sistemov pa terja tudi združitev obeh vrst komponent z namenom, da pridemo do kompletne slike generalizirane ekonomike. Pri analizi posameznih komponent evaluacije se običajno zadržimo pri strukturnem modelu ekonomike direktnih stroškov informacijskih procesov in informacijskih sistemov /1/. Naloga je, poiskati podobne modele tudi za druge sfere, ki niso ekonomske, tako za stroškovne kot tudi za vrednostne komponente, pa v doslej znani literaturi ne pozna analogij k funkcionalnemu modelu, kot je na primer strukturni model ekonomike direktnih stroškov.

V naslednjem obravnavamo sintezo generaliziranih stroškovnih in vrednostnih komponent; njihovo unijo označimo z $I(t)$ kot skupnost evaluacijskih komponent in jo imenujemo multidimenzionalni evaluacijski tok informacijskega sistema. Naj bodo vse komponente evaluacijskega toka $I(t)$ informacijsko merljive na vsaj en način (ki je lahko tudi čista konvencija). Pri tem v skladu z input-output orientacijo tudi tu v splošnem predvidevamo, da je evaluacijski tok diferenciran glede na vhod, izhod in stanje v informacijskem sistemu. Poudarjamo, da je evaluacijski tok množica procesov, ki jih kot koinformacijske procese (npr. ekonomske, organizacijske, itd.) pripisujemo informacijskemu sistemu in je pri tem ta množica inducirana s strani osnovne množice informacijskih procesov. Predpostavimo v nadaljnjem, da je tokovna komponenta $p_1(t) \in I(t)$ merljiva tudi na vsakem procesnem mestu (npr. organizacijskem, stroškovnem, itd.) $E_1, \ldots E_m$. Takšno procesno mesto lahko imenujemo evaluacijsko mesto. Naj bo v nadaljnjem razmišljanju takšnih evaluacijskih centrov toliko, kot je procesnih mest. Izhodno okolje pripišimo procesnemu mestu $E_{(i)}$ in podobno vse output evaluacije pripišimo procesnemu elementu $E_{(o)}$ in pripadajoča dela evaluacijskih tokov označimo zaporedoma z $P^{(i)}(t)$ oz. $P^{(o)}(t)$. Naj bo $\hat{I}(t)$ zvezni evaluacijski tok, ki pripada $I(t)$ in naj vsebuje komponente $\hat{p}_1(t), \ldots, \hat{p}_n(t)$. Vhodni evaluacijski tok $\hat{p}^{(i)}$ $\hat{X}(t)$ ima strukturo $\hat{x}(t)=(\hat{x}_1(t),\ldots\hat{x}_n(t))' \in \hat{X}(t)$ in je alociran k množici procesnih mest v prvem koraku s pomočjo alokacijskega operatorja $A_1(t,t_1)$. Tako dobimo distribuirani evaluacijski input pri prvem koraku

$$A_1(t,t_1) \begin{pmatrix} \hat{X}_1(t) \\ \vdots \\ \hat{X}_n(t) \end{pmatrix} = Z^{(1)}(t,t_1)$$

Pri drugem koraku, ki npr. pripada času $t+t_1+t_2$ imamo naslednjo alokacijo

$A_2(t,t_1,t_2)\hat{Z}^{(2)}(t,t_1,t_2)$ itd., dokler ne dosežemo output evaluacije

$$A_\sigma(t,t_1,\ldots t_\sigma)A_{\sigma-1}(t,t_1,\ldots t_{\sigma-1})\ldots A_1(t,t_1)\begin{pmatrix}\hat{x}_1(t)\\ \vdots\\ \hat{x}_n(t)\end{pmatrix} = \begin{pmatrix}\hat{y}_1(t,t_1,\ldots t_\sigma)\\ \vdots\\ \hat{y}_N(t,t_1,\ldots t_\sigma)\end{pmatrix} \qquad (2)$$

pri čemer ima vsak $\hat{y}_j(t,t_1,\ldots t_\sigma)$ dimenzijo $N \lessgtr n$. Zgornja relacija dovoljuje a) proceduro cepitve evaluacijskih tokov, $N>n$; b) proceduro agregiranja evaluacijskih tokov, $N<n$; c) proceduro, ki je invariantna glede na začetno evaluacijo $\hat{x}(t)$. Za informacijski sistem splošne narave očitno velja $t_1=t_1(t)$, $t_2=t_2(t)$, $t_\sigma=t_\sigma(t)$, $\sigma=\sigma(t)$, ki so slučajne funkcije. Če definiramo $A_o=I$, $A_{\sigma+1}=$ I kot identična operatorja, imamo celotni kompleks $\hat{I} = \hat{X}_x\hat{Z}_x\hat{Y}$ strukturiran glede na informacijski sistem, t.j.

$$\hat{p}(t)=(A_o(t),A_1(t,t_1),\ldots A_{\sigma+1}(t,t_1,\ldots t_{\sigma+1}))\hat{x}(t) \qquad (3)$$

Vsako procesno mesto utegne biti "dotaknjeno" pri kakšnem t in t_p, p=1,...σ+1. Tako $\hat{x}(t)$ postane pretok (throughput) pri kakem končnem σ+1. Vzemimo sedaj kak element $\hat{p}_j(t)$ v njegovem p-tem koraku

$A_r(t,t_1,\ldots t_p)\ldots A_1(t,t_1)\hat{x}_j(t) = z_j^{(p)}(t,t_1,\ldots t_p)$

ki opisuje lokacijo k E_i, t.j. $z_{ji}^{(p)}(t,t_1,\ldots t_p)$.

Vsak $\hat{x}_j(t)$ je vektor; takšen je tudi $\hat{Z}_{ji}^{(p)}$ za V_p, V_i. Če $A_{p+1}(t,t_1,\ldots t_{p+1})$ ima koeficient enote pri tistem delu, ki se nanaša na i in j, potem $Z_{ji}^{(p)}(t,\ldots t_p)$, ki je bil input za E_i, v p-tem koraku postane output za E_i v (p+1)-tem koraku. Vzemimo, da smo to evaluacijo prenesli k $E_{i'}$. Če je takšna evaluacija tista komponenta, ki spada v ekonomiko stroškov, potem tak transfer poznamo kot transakcijo. Tu bi za splošni primer lahko definirali

$$A_{p+1}\hat{Z}_{ji}^{(p)}(.)-\hat{Z}_{ji}^{(p)}(.)= \hat{Z}_{ji}^{(p)}(.) \qquad (4)$$

kot bilanco za $Z_{ji}^{(p)}(.)$, ki pripada (p+2)-terici časovnih zapisov $t,t_1,\ldots t_{p+1}$.

Relaciji (3) in (4) nam povesta, da je output v smeri od E_i enak inputu za $E_{i'}$. Za vsako procesno mesto E_i, V_i, definiramo dva evaluacijska tokova, input in output tok (označena z + in -).

V splošnem imamo tri primere za mero vsakega evaluacijskega vektorja: 1) mera zavisi od j; 2) mera zavisi od i; 3) mera zavisi od p. Ker smo izbrali $\hat{z}_{ji}^{(p)}(.)$ v vektorski obliki, nastopa tudi vprašanje, ali 4) ta mera zavisi tudi od k.

Predvsem se moramo ozirati na odvisnost te mere od k, kar pomeni, da se moramo ukvarjati z množico mer $n_1,\ldots n_k$ na temelju enclične in obratno lične korespondence. Kar zadeva prvi primer zgoraj, lahko vedno definiramo maksimalno množico mer tako, da za vsak j obravnavamo tako povečano množico mer na ta način, da postavimo mero enako nič, če le-ta ne eksistira za kak posamezen j. S formalnega vidika smo se torej izognili težavi, ki jo prinaša variabilnost mere glede na j. Na isti način lahko ukrepamo tudi v primeru (3), če le definiramo maksimalno množico mer, ki pripadajo vsem p in potem postavimo mero enako nič, če za kak korak p ta mera ne eksistira. Vendar pa se moramo zavedati, da v obeh zgornjih primerih te ničle pomenijo "logične" ničle, ki niso rezultat ničelnih veličin. Primer 2) utegne biti težaven, kar je povsem realen primer. Pokažemo lahko, da ta problem lahko postopno vključimo v pristop, ki je primeren za obravnavanje primera 4).

Da bi lahko prikazali osnovno idejo informacijskega modeliranja multidimenzionalne evaluacije informacijskih sistemov, se bomo osredotočili za hip na kako izbrano mero μ_k. Označili jo bomo kot dekomponibilno na paru E_i in $E_{i'}$, če eksistira na vsakem od teh procesnih mest. To torej pomeni, da je k-ta komponenta merljiva z μ_k pri dveh procesnih mestih. Ta definicija pa se dá raztegniti na vsako podmnožico procesnih mest. Vendar pa v praksi lastnost dekomponibilnosti ne velja za vsak k. Nadalje, μ_k v splošnem ne izpolnjuje zakona o ohranitvi bilance, kot je npr.

$$\mu_k(\Delta\hat{Z}_{ji,k}^{(p-1)}(.))+ \mu_k(\Delta\hat{Z}_{ji,k}^{(p)}(.) = o \qquad (5)$$

Prav tako pričakujemo, da bi tudi vrednostne mere kot mere, uporabljene v multidimenzionalnem evaluacijskem modelu, morale izpolnjevati zahtevo po ohranitvi zakona vrednosti. V teh dveh primerih μ_k poseduje lastnost tranzitivnosti. Pravimo, da je ta mera tranzitivna glede na $(E_i, E_{i'})$, če se pripeti posamična transformacija $\hat{Z}_{ji,k}^{(p-1)}(.) \rightarrow \hat{Z}_{ji,k}^{(p)}(.)$ tako, da je mera μ_k dekomponibilna preko E_i in $E_{i'}$, ter izraža lastnost tranzitivnosti; takšno transformacijo lahko imenujemo transakcijo.

Spomnimo se, da celotna množica {p} utegne pripadati kakemu specialnemu i; da bi poenostavili našo predstavitev, bomo dovolili samo eno vrednost za p pri vsakem i ; zato bomo indeks p v nadaljnji pisavi

$$\hat{Z}_{ji,k}^{(p)}(.)$$

opustili. Predpostavimo tudi, da imamo vse $\hat{Z}_{ji}(.)$ dvojne narave: imajo transaktivne kot tudi ne-transaktivne mere (za različne k). Odtod sledi, da je transformacija lahko transakcija ali pa ne-transakcija. Osredotočimo se samo na transakcije, kjer je k=1 za kak k. Pri predpostavki o izpolnjenosti zakona o ohranitvi (nasploh), iz (5) sledi

$$\mu_k(\Delta\hat{Z}_{jik}(.) = -\mu_k(\Delta\hat{Z}_{jik}(.)) \qquad (6)$$

če le predpostavimo enolično korespondenco $p \rightarrow i$. Če je mera $\Delta\hat{Z}_{jik}(.)$ pozitivna za E_i, potem je ta mera negativna za $E_{i'}$. Prostor vseh možnih transakcij lahko v tem primeru opišemo z množico matrik tipa

$$\Omega_{ii'}^{jk} = \begin{pmatrix} o & \ldots & o & \ldots & o\\ \cdot & \cdot & \cdot & \cdot & \cdot\\ o & \ldots & \omega_{ii'}^{jk} & \ldots & o\\ \cdot & \cdot & \cdot & \cdot & \cdot\\ o & \ldots & o & \ldots & o\end{pmatrix} \qquad (7)$$

na temelju (6), za katero so absolutne vrednosti

$$\left|+\mu_k(\Delta\hat{Z}_{jik}(.))\right| = \left|-\mu_k(\Delta\hat{Z}_{jik}(.))\right| = \omega_{ii'}^{jk}.$$

To matriko bomo imenovali transakcijski impulz med E_i in $E_{i'}$ (od E_i k $E_{i'}$). Na ta način lahko (7) imenujemo impulzno matriko; le-ta nam kaže transakcijo med E_i in $E_{i'}$, znesek te transakcije, $\omega_{ii'}^{jk}$, pa je sprememba v k-ti komponenti j-tega procesa. Ker je dana korespondenca med p in i, je vsakemu paru (i,i') določen par (p,p'). Med obračunskim razdobjem, v katerem obravnavamo ekonomiko informacijskega sistema v posplošenem pomenu besede, lahko pričakujemo transakcijske impulze $\omega_{ii'}^{jk}$, . Da bi vzdrževali enolično evidenco takšnih impulzov, moramo dodati še časovni indeks, torej $\omega_{ii'}^{jk}(t)$. Na ta način impulzna matrika $\Omega_{ii'}^{jk}(t)$ postane dinamizirana. Celotni informacijski tok k-te komponente j-tega procesa je v obravnavanem razdobju podan kot $\sum_t \sum_{\{t_i\}} \Omega_{ii'}^{jk}(t) = \Omega^{jk}$. S tem pa je

$$((1,\ldots 1),\ldots(1,\ldots 1))\Omega^{jk} = (\omega_o^{jk,-},\ldots \omega_{s+1}^{jk,-}) = \Omega^{jk,-} \qquad (8)$$

ki je vektor totalnega evaluacijskega input toka za procesna mesta $E_{(i)},\ldots E_{(s+1)=E(o)}$. Podobno lahko definiramo

$$\Omega^{jk}\begin{pmatrix}1\\ \vdots\\ 1\end{pmatrix} = \begin{pmatrix}\omega_{(i)}^{jk,t}\\ \omega_1^{jk,t}\\ \vdots\\ \omega_{s+1}^{jk,t} = \omega_{(o)}^{jk,t}\end{pmatrix} = \Omega^{jk,t} \qquad (9)$$

kot vektor totalnega evaluacijskega output toka za isto procesno mesto. Odtod pa je $\Omega^{jk,t} - (\Omega^{jk,-}) = \Delta\Omega^{jk}$

posplošeni bilančni vektor. Na ta način smo vključili tudi klasično ekonomiko, saj za transaktivne mere evaluacije informacijskega sistema veljajo vse lastnosti klasičnega knjigovodskega sisema. Naš namen pa je seveda vključevati tudi druge evaluacijske komponente. Če je μ_k transaktivna mera, potem se da hitro pokažati, da smo kos knjigovodskemu konceptu evaluacije informacijskega sistema z uvedbo običajne matrične algebre na skalarnih komponentah. Če pa vzamemo drugačne mere od transaktivnih, potem za posamezne evaluacije komponente potrebujemo primarne algebre; le-te nam omogočajo spremljanje transformacij po posameznih korakih za isto evaluacijsko komponento. Primer transaktivne mere kaže po eni strani možnost vključevanja klasične ekonomike v generalizirano evaluacijo informacijskih sistemov, po drugi strani pa pot za konstrukcijo algeber, ki pripadajo ne-transaktivnim meram. Množica različnih ko-informacijskih procesov je opisana z množico matrik Ω^{jk} in ustreznimi algebrami. Predpostavimo, da dani informacijski sistem lahko razbijemo v končno mnogo organizacijskih enot, npr. $U_1,\ldots U_j,\ldots U_q$, ki so agregati procesnih mest; pri tem je vsaka enota opisana z ustrezno

Ω^{jk}, j=1,...q. Za dani informacijski sistem imamo s tem sintetično predstavitev

$$\Omega^{(k)} = \begin{pmatrix}\Omega^{1k} & & \\ & \Omega^{jk} & \\ & & \Omega^{qk}\end{pmatrix}$$

kjer izven diagonale vzdržujemo matrike, ki nam kažejo "interface" med organizacijskimi enotami (strukturiranimi s ko-informacijskimi procesi). Če ni nobenih medsebojnih tokov med enotami, potem je informacijski sistem opisljiv na področju prostora ko-informacij z med seboj ločenimi multidimenzionalnimi evaluacijskimi modeli. Vsak končen nabor matrik $\Omega^{(k)}$ bomo tedaj imenovali informacijski model multidimenzionalne evaluacije danega informacijskega sistema. Tako koncipiran evaluacijski model podaja distribuirano sliko evaluacijskega toka I(t) po vsem informacijskem sistemu.

/iri

(1) V.Rupnik - G.Resinovič - J.Grad, Ekonomika informacijskih sistemov; FCI - EFBK, raziskovalna naloga po naročilu CAOP - Iskra - ZORIN; Ljubljana, 1985.

MJESTO I ULOGA PERSONALNIH RAČUNARA U INFORMACIJSKOM SISTEMU PRIVREDNE ORGANIZACIJE

Goran Koch,
INA - PETROKEMIJA OMIŠALJ
Omišalj, SFR Jugoslavija

UDK: 681.3.007

SAŽETAK

Personalna računala su vrlo brzo nakon svoje prve pojave postala veoma interesantna i za upotrebu u privrednim organizacijama. Razlog tome jest u izuzetno povoljnom odnosu niske cijene i visokih perfomansi čime se postiže njihova vrlo brza aktualizacija. U radu se razmatraju, sa stanovišta privredne organizacije tri tipična modela upotrebe. Prvi model govori o mogućnosti upotrebe personalnog računara za zadovoljavanje "klasičnih" informatičkih potreba masovne obrade knjigovodstvenih i drugih podataka. U drugom modelu se ukazuje na visoku efikasnost personalnih računara upotrijebljenih kao samostalne jedinice CAD sistema ili informatizirane visokostručne radne stanice, uz korištenje integriranog software-a uključujući elemente umjetne inteligencije. Kroz treći model se opisuje povezivanje personalnog računara u mrežu s već postojećim centralnim sistemom. Dan je prikaz moguće upotrebe posebno s osvrtom na predrosti personalnog računara kao satelita u mreži i ukazano je na pojavu trodomne strukture podataka: centralna baza podataka, lokalni podaci i privatni podaci. Zaključno se iznose glavni razlozi najavljivanja da će personalni računari vrlo brzo postati bitnim elementom informacijskih sistema privrednih organizacija.

SUMMARY

Soon after their first appearance, personal computers become very interesant for use in business organisation. Reason for that is in exceptionally profitable relationship between low price and high performances by which their fast actualisation is achieved. Three typical models of usage in the business organisation are discussed in the paper. First model speeks about possibility of using personal computer for doing "classical" informatical jobs as are mass data processing. In the second model, paper points at high efficiency of personal computers used as CAD stations or informatized expert workstation including integrated software and elements of artificial intelligence. Through third model, personal and centralised host computer network is discussed. Description of possible usage, with considering of preferences of personal computer satellite, is given. It is pointed at three level data structure: central data base, local data and private data. At the end, there are given main reasons for announcements that personal computers will soon become the most important elements of information systems in business organizations.

1. O FENOMENU PERSONALNIH RAČUNARA

Uobičajeno je da se u dosadašnjem razvoju računarske opreme razlikuju četiri tzv. generacije računara (uz najavljenu skoru pojavu pete). Prvu generaciju u načelu karakterizira upotreba elektronskih lampi u sklopovima; drugu pojava i upotreba tranzistora; treću tehnologija integriranih krugova a četvrtu LSI i VLSI tehnologije vrlo visokog stupnja integra-

cije. Uz ove tehnološke razlike možemo uočiti i očite razlike reda veličine centralne memorije: od stotinjak memorijskih lokacija preko reda veličine nekoliko K (1024) byte, odnosno nekoliko stotina Kb do današnjih veličina od nekoliko M (mega, milijuna) byte. Jednako tako se mijenjala i tipična konfiguracija i namjena računara: od sofisticiranih "čudovišta visoke znanosti" preko prvih nezgrapnih mašina opće upotrebe do računara manjih dimenzija i velike snage u centraliziranoj konfiguraciji i do današnjih velikih sistema dislocirane opreme, mreže računara i terminala.

Sredinom sedamdesetih godina se međutim u tehnološkom razvoju pojavio mikroprocesor, kompletan računar (CPU) unutar jednog elektroničkog elementa. Kontroverzije oko fantastičnih mogućnosti upotrebe a posebno strahoviti tržišni boom personalnih i kućnih računara početkom 80-tih godina (tipično: Apple i ZX), upozorili su na njihovu opću važnost. Glavne prednosti zahvaljujući kojima personalni računari osvajaju informatički prostor jesu njihova mala cijena naspram potpunih računarskih mogućnosti. Tipična konfiguracija dosiže centralnu memoriju 64 - 128 K byte, disk jedinicu 10 M byte (i/ili adekvatnu jedinicu disketa), kolor moritor te matrični printer za cijenu reda veličine 2.000 - 5.000 USA $.

U brzom razvoju personalnih računara jasno se mogu uočiti dvije njihove generacije: 8-bitni računari prve generacije (Apple II,IIe,ZX,CBM 64, Partner, Ivel Z-3) te 16-bitni računari druge generacije (IBM PC, Apple : Macintosh, CBM PC 10, Sperry PC, ZX - QL itd.). Najavljena je i treća generacija 32 bitnih računara koji bi se trebali masovnije pojaviti krajem ovog desetljeća.

Prva od ove tri generacije je bazirana na mikroprocesorima 6502, 6802 odnosno Z 80 a druga uglavnom na Intel 8088 odnosno (sve više) Motorola 68000. U, nažalost šarolikom, svijetu software podrške, može se kao najmasovniji operativni sistem kod 8-bitnih računara prepoznati CP/M a kod 16-bitnih MSDOS (PC DOS) dok je programski jezik BASIC zastupljen bez iznimke kod svih.

2. PRIMJENA PERSONALNIH RAČUNARA

Uz tehnički faktor (CPU na jednom čipu), postojanje personalnih računara omogućio je i ekonomski faktor (niska cijena) koji su zajedno pronašli tržište na području gdje ga do tada osim u SF romanima nije bilo. Računar je odjednom doživio svoju krajnju točku razvoja prema korisniku: postao je lično sredstvo za rad poput olovke i papira. Upravo ta personifikacija upotrebe je otvorila vrata kako novoj proizvodnji i tehnološkoj grani tako i znanstveno - tehničkim rješenjima. Današnja personalna računala druge generacije već pokazuju ambiciju da premoste jaz prema velikim sistemima te tako omoguće potpunu integraciju svih AOP funkcija. U tom smislu tipično personalno računalo može tehnički biti upotrijebljeno na jedan od slijedećih načina:

- kao samostalna jedinica - personalni računar
- kao element mreže personalnih računara
- kao inteligentni terminal velikog centralnog računara
- kao satelitski računar u mreži s velikim centralnim računarom
- kao kombinacija prethodnih načina.

Već sama ova podjela ukazuje na "proizvodjačku" tendenciju razvoja koja nastoji i personalne računare "ugurati" u integralne sisteme AOP. Time se s jedne strane brže postiže prijeko potrebna standardizacija a s druge strane se direktnije ukazuje na potrebu izgradnje globalnih komunikacijskih sistema.

Nas će ipak zanimati jedan drugi aspekt upotrebe personalnih računara: korisnički i to specijalno sa stanovišta privredne organizacije. U tom smislu valja odmah primjetiti da bez obzira na početne (i još prisutne) pokušaje minorizacije upotrebe personalnih računara u privredi, njih je ipak nemoguće izbjeći. Za takvo što (minorizacija) njih već sada ima previše, imaju preveliku informatičku snagu i očite su tendencije sve šire upotrebe personalnih računara ne samo za "lične" ili "kućne" obrade podataka i programiranje igara, već i za stvarne poslovne obrade sve od masovnog zahvata podataka pa do poslovnih simulacija.

S korisničkog, privrednog, stanovišta očita su tri karakteristična modela upotrebe personalnih računara: model potpune informatizacije samostalnim računarom; model specijalizirane lokalne visokostručne upotrebe; te model upotrebe personalnog računara kao dijela mreže računara u pokrivanju jednog dijela mogućeg spektra informatizacije poslovanja. Pokušat ćemo ukratko opisati bitne značajke svakog od tih modela.

MODEL 1: Personalni računar je u stanju da zadovolji sve informatičke potrebe manjih privrednih organizacija. U ovom modelu upotrebe je tipičan interes da se računarom relativno niske cijene automatiziraju svi "klasični" poslovi prvenstveno knjigovodstva, materijalnog i financijskog. Pri tome je direktna ideja ta da se naprosto preslika logika pristupa AOP gdje se umjesto skupog "pravog" računara upotrebljava personalni.

Obzirom na širok zahvat velike količine ulaznih podataka u takovoj ideji upotrebe, odmah je vidljiv glavni ograničavajući faktor modela: relativna sporost unosa podataka. Uzimajući u obzir standardne normative rada treniranih operatera, sistemske potrebe izgradnje i održavanja programa, te potrebe održavanja samog računara, dolazimo do brzine unosa od oca 10 000 znakova dnevno. Pri tome gotovo da i nema mjesta povećanju ovog broja naprosto zbog toga što je ograničenje tehničkog karaktera. U organizacionom smislu bi se moglo povećati kapacitet unosa podataka na način da se uvede rad više smjena. Može se procijeniti da uvodjenjem druge smjene (pri čemu se ona gotovo u potpunosti koristi za unos podataka) raste kapacitet unosa podataka na oca 50 000 znakova.

Takodjer je moguće tražiti povećanje ulaznog kapaciteta u odredjenim tehničkim rješenjima ukoliko se problemu pristupi kroz nabavku više personalnih računara. Tada je moguće ili njihovo "on line" povezivanje u mrežu ili "off line" povezivanje putem razmjene podataka na lako prenosnim disketama. Po logici upotrebe, svaki se personalni računar tada upotrebljava lokalno za pokrivanje odredjene poslovne cjeline (npr.: kadrovi i osobni dohoci, financijsko knjigovodstvo, skladišno-materijalno poslovanje itd.).

Upozorimo, da ukoliko se i poveća kapacitet unosa podataka, odmah nastupa drugi ograničavajući faktor: mogućnosti papirnatih izlaza. Personalni računar je namijenjen prije svega (po svojim tehničkim osobinama) izlazu podataka putem ekrana uz odredjene grafičke mogućnosti pri čemu matrični printer služi samo za sporadične ispise. Masovne štampe klasičnih "izvještaja" nisu realno ostvarive.

Očito je dakle da uz atraktivnost cijena personalnog računara, postoje vrlo realna ograničenja njegove konkretne upotrebe. Ova ograničenja svode model i u okvire interesantnosti tek za male ili srednje privredne organizacije reda veličine do oca 500 zaposlenih.

MODEL 2: Personalni se računar može maksimalno efikasno koristiti upravo u okvirima svog naziva: kao lično sredstvo za rad. Ovaj model upotrebe predvidja osnivanje specijaliziranih visokostručne potpuno automatiziranе radne lokacije kojom se koristi ograničeni broj (idealno jedan) radnika. Tipične su tzv. CAD (Computer Aided Design) informatičke radne stanice. U informatičkom smislu osnovne karakteristike ovakve upotrebe personalnog računara jesu relativno mala količina ulaznih podataka, veoma složene naučno-tehničke programske aplikacije te potreba za grafičkim izlazom visoke kvalitete. Osnovnoj konfiguraciji računara će u ovakvom radu biti potrebno dodati i crtač krivulja (plotter) odnosno ekran visoke razlučivosti (tipično 1024x1024) adresibilnih točaka).

Možemo slobodno reći da ovaj model upotrebe do krajnosti ističe sve povoljne karakteristike personalnih računara. Tehnički problemi konstrukcije automobila, aviona, cipela, plakata, programa, informacijskih sistema, praktično bilo čega, se na taj način u potpunosti rješavaju. Ovaj model je zato naročito pogodan u projektnim i razvojnim odjelima privrednih organizacija i ujedno se ekonomski najbrže i najviše isplati.

Varijanta upotrebe personalnih računara ide naravno za time da ovakovim modelom rada unaprijedi i stručne poslove ne-tehničkog karaktera (ekonomski, pravni itd.). Ovdje istaknuto mjesto zazuzimaju integrirani programski paketi koji se redovito sastoje od 4 dijela: privatna baza podataka, tekst-procesor, kalkulator te grafički prikaz rezultata. Povećanje efikasnosti "administrativnog" rada je svakako primamljiv cilj svakoj privrednoj organizaciji a ovo je idealni model upotrebe računara za dosizanje tog cilja.

Ne smije se u ovom modelu zanemariti jedan pravac razvoja primjene personalnih računara koji će vrlo brzo ukazati na nove puteve saradnje čovjeka i stroja: na tehnike tzv. "umjetne inteligencije". Ekspertni sistemi su već stvarnost, a u eksploziji informacija u svijetu, jedino sistemi sposobni da uče jesu odredjena budućnost. Možemo prihvatiti procjene da će posebno u ovom modelu upotrebe,

tehnike umjetne inteligencije, sistema autokontroliranog rasta, samoučeći eksperti, odnijeti prevagu već u narednih 2 - 3 godine.

MODEL 3: U privrednim sistemima koji već posjeduju računare III ili IV generacije sa ostvarenom (ili mogućnošću ostvarenja) mreže dislociranih terminala, moguće je iskoristiti personalno računalo na način da ostvaruje povoljne karakteristike oba dosad rečena modela uz još i dodatne prednosti. Personalni računar je upravo idealno satelitsko računalo u mreži: relativno niska cijena, malene fizičke dimenzije, širok dopustivi dijapazon uvjeta rada. Uza sve, to je ujedno i visokosposobni računar.

U jednom takvom sistemu, u dijelu masovne obrade podataka, personalni računar preuzima ulogu inteligentnog terminala putem kojeg se obavlja:

- ulazna kontrola podataka, sortiranje i formiranje datoteka te komunikacija s računarom putem transfera datoteka a ne podataka,
- izlazni prikaz podataka putem grafike visokog nivoa primjenjivosti, koja ne opterećuje glavni računar,
- komunikacija s operaterom-korisnikom na "inteligentnom" nivou ekspertnog sistema običnim jezikom dotične struke.

Personalni računar se pokazuje u ovakvom modelu upotrebe kao idealna, ekonomski izuzetno opravdana, mogućnost ostvarenja ideje distribuirane obrade podataka. Ujedno, kroz mrežu računara se na ovaj način tehnički izuzetno dobro podržavaju integralni informacijski sistemi. Mreža satelitskih računara omogućava potpunu kontrolu odabira nivoa centralizacije i lokalizacije obrade podataka odnosno optimizacije podataka u transportu kroz mrežu.

Personalni računar i nadalje obavlja samostalne funkcije CAD sistema uz dodatnu prednost da sve veće probleme koji iziskuju snaženja resursa može rješavati transferom obrade na centralni računar. To znači da područje upotrebe personalnog računara možemo proširiti i na složene numeričke probleme npr. diskretne i kontinuirane simulacije. Na taj način se stvara jedna specifična simbioza centralnog računara velike snage kojemu se maksimalno koriste sposobnosti kao što su brzina i veličina memorije sa personalnim računarom kojim se maksimalno unapređuje (prema grafici i umjetnoj inteligenciji) ulazno - izlazna komunikacija s čovjekom.

Uz navedene opće pretpostavke, u ovom se modelu korištenja javlja i specifična trodomna organizacija podataka u računskoj mreži:

- baza podataka na centralnom računaru koja sadrži sve osnovne podatke relevantne za ukupni, integralni informacijski sistem,
- lokalne datoteke na centralnom računaru koje sadrže podatke velikog obima ali uskog lokalnog značenja ili podatke u transferu kroz mrežu,
- privatne datoteke ili čak baze podataka smještene na eksternim memorijama personalnih računara a pod neposrednom kontrolom korisnika, sa podacima manjeg obima a većeg stupnja tajnosti ili privatnosti.

3. ZAKLJUČAK

Personalni računari u privrednoj sredini su očigledno izazov i interes. Pomoću njih je moguće (ukoliko ih se adekvatno upotrijebi) konačno u potpunosti premostiti jaz između ljudi i automatske obrade podataka.

Ujedno, personalni računari su za privredu jedna velika šansa. Period njihove aktualizacije (ekonomske isplativosti investicije) veoma je kratak, tipično 6 mjeseci do godine dana. Taj period je ovoliko kratak zbog istovremenog djelovanja dva faktora: niske cijene te široke mogućnosti neposredne operativne upotrebe.

I na kraju, potrebno je ukazati na jednu činjenicu o kojoj nije bilo eksplicitno do sada rečeno ništa ali: personalni računari će najdirektnije ukazati da su posebno u informatici, ljudski resursi najdragocjeniji. Šansu, i to veliku, imaju one privredne sredine koje će biti sposobne da iskoriste vlastite kadrovske stručne i znanstvene potencijale. Upravo zbog tog odnosa će personalni računari postati bitnim elementom informacijskih sistema privrednih organizacija već koncem ovog desetljeća.

METODOLOŠKI PRISTAP VO PLANIRANJETO NA INFORMACIONI SISTEMI VO UPRAVA

Mirjana Apostolova, Vida Trajkovski, Gradski zavod za organizacija i razvoj na Informacionen sistem, Skopje,SFRJ

UDK: 681.3.007

Referatot gi prikažuva iskustvata od doslednata primena na principot "koncepcijata i razvojot na Informacioniot sistem da ja planiraat onie, koi što utre i direktno će ja koristat." Osnoven akcent e daden na organizacioniot aspekt od sprovedenata "Studija i plan za realizacija na Informacioniot sistem na Grad Skopje", a voedno i rezultatite i iskustvata dobieni vo tek na izgotvuvanje na Studijata, od aspekt na analizata na postojnoto rabotenje na organite na upravata, projavenite problemi, iskustvata vo utvrduvanje na osnovnata koncepcija na Informacioniot sistem i negovata konkretna arhitektura.

METHODOLOGICAL APPROACH FOR PLANNING OF THE INFORMATION SYSTEM IN LOCAL GOVERNMENT

There, we presents some experiances in using the princip "planning and development of the information syustem to be made from someone, who will use it tomorrow"... not from EDP profesionals. The accent is put on the organizational aspect of the realizated "Study and plan for realization of the information system, results and experiences from using IBM methodology - "BSP - Bussines System planning". The end - users formulated their information system plans and control mechanisms to improve their use of information and data processing resources. We are persuaded that the first and most important objective of BSP is to provide an information system plan that supports the businesss short - and long - term information needs and is integral with the business plan.

Kako da se pristapi kon planiranjeto na informacionite sistemi? - voobičaena e dilema pred koja se naogja skoro sekoj EOP centar od kogo se bara do utre da predloži: koncepcija na informacioniot sistem da gi definira prioritetite, da ja utvrdi potrebnata konfiguracija i da gi isplanira potrebnite resursi.

Gradot Skopje pri planiranjeto na Informacioniot sistem se opredeli za primena na principot na metodološki pristap vo planiranjeto i realizacijata na Informacioniot sistem koj pretstavuva edinstven način da se obezbedi dobra komunikacija na site učesnici vo planiranjeto i realizacijata na Informacioniot sistem i poširoko.

Pri izborot na metodološkiot pristap se odlučivme na standardnata metodologija koja ja preporačuva IBM, so opredeluvanje da vo tek na primenata ja usoglasuvame so konkretnite uslovi i potrebi. Metodologijata za planiranje na Informacioniot sistem se bazira na IBM - ovata metoda "BSP - Business system planning," čii osnovni karakteristiki se:

- planiranjeto na Informacioniot sistem e zadača na Studiski tim sostaven od rakovodni kadri na instituciite koi se direktni učesnici i korisnici na Informacioniot sistem,
- razvojot na Informacioniot sistem se smeta kako integralen del na kratkoročnite i dolgoročnite planovi za razvoj,
- obezbeduva identifikacija na podatocite i aplikativnite sistemi, utvrduva prioriteti vo razvojot i garantira usoglasenost na zaedničkite resursi na idniot Informacioniot sistem, i
- obezbeduva edinstveno planiranje i koristenje na osnovnite resursi na Informacioniot sistem i nudi sigurnost ne samo na rakovodnite strukturi tuku i na krajnite korisnici vo ispravnosta na ponudenite rešenija i opravdanost na planiranite investicii.

Vo ovoj referat se izneseni iskustvata od primenata na BSP metodologijata so osnoven akcent na organizacioniot aspekt od sprovedenata "Studija i plan za realizacija na Informacioniot sistem" i rezultatite dobieni vo tek na nejzinoto izgotvuvanje.

I FAZA - PODGOTOVKI

So starešinite na organite i instituciite koi bea opredeleni kako prioritetni korisnici na Informacioniot sistem, održan e dogovoren sostanok so cel točno da se utvrdat rezultatite što treba da se dobijat od Studijata da se definira tim koj će ja realizira Studijata i načelno se dogovorat odnosite pomegju subjektite. Za členovi na timot glavno bea izbrani rakovodni rabotnici so dolgogodišno organizaciono iskustvo i so solidno poznavanje na rabotnite procesi vo svojot domen na rabotenje. Istaknato beše deka za realizacija na Studijata će se koristi konsultativna pomoš od "Intertrade" zaradi dosledno sproveduvanje na metodologijata za planiranje na informacionite sistemi. Isto taka izgotveno e globalno terminiranje na aktivnostite i utvrdeno e deka so maksimalno angažiranje na site učesnici Studijata treba da se izgotvi za dva meseca. Opredeleni se mentor na Studijata, rakovoditel na Studiskiot tim i tehnički sekretar. Karakteristično e da se istakne deka vo timot edinstveno rakovoditelot beše "EOP profesionalec". Od tie pričini postoeše izvesno somnevanje kaj drugite EOP profesionalci

deka vakov konstituiran tim ne e dorasnat na postavenata zadača. Bea izvršeni i drugi podgotovki kako što se: utvrduvanje na obemot i sodržinata na konsultativnata pomoš, napraveni se detalni planovi na aktivnosti za sekoja faza, izgotven e kostur (ramka) na završniot izveštaj i se obezbedija site uslovi za nesmetano rabotenje na timot.

II FAZA - ŠKOLUVANJE

Osnovna zamisla ni beše členovite na Studiskiot tim, paralelno so izgotvuvanje na Studijata, da go nadograduvaat svoeto znaenje od oblasta na EOP i da se zapoznaat so metodologijata koja će ja primenuvaat i na koja će rabotat. Za taa cel organiziravme poseta na kursot koj ja obrabotuva metodologijata za planiranjeto na informacioni sistemi, a voedno organiziravme seminari na koi pokaneti predavači govorea na poveće temi.Vo tek na izgotvuvanjeto na Studijata organiziravme nekolku posети na po oddelni centri pri što beše razgovarano so rakovoditelite na centrite se so cel "vo živo" da se zapoznaat so kompjuterot i negovite možnosti da se razmeni iskustvo i da se proverat postavkite do koi beše dojden timot. Vaka organizirana priprema dade rezultat. Za kratko vreme členovite na Studiskiot tim se vklučija vo rabota, go prenebregnaa kompleksot što go čustvuvaa prema EOP i svatija deka tie treba da bidat nositeli na planiranjeto na idniot razvoj na Informacioniot sistem.

III FAZA - ANALIZA NA POSTOJNOTO RABOTENJE

Za otpočnuvanje so rabota na timot i megjusebno zapoznavanje, se organiziraa prezentacii vo koi sekoj člen na studijskiot tim ja prezentiraše svojata institucija, delokrugot na rabota i osnovnite problemi so koi se soočuvaat, isto taka se izgotvi terminološki rečnik koj kako iskustvo se pokaža mnogu korisno vo razbiranje na stručnata terminologija koja ja koristeše sekoj člen vo timot od svojata oblast a voedno i zaradi razbiranje na EOP terminologijata. Analizata na postojnoto rabotenje se odvivaše preku utvrduvanje na osnovnite celi, proizvodi i resursi na sekoja od analiziranite institucii i opredeluvanje na glavnite rabotni procesi i osnovnite klasi na podatoci. Od praktični pričini, za ovie kategorii se počituvaše ograničuvanje na analizata na "prvite deset" bidejći postoeše opasnost da se navleze vo nepotrebni detali. Kako rezultat od izvršenata analiza se definiraa glavnite rabotni procesi utvrdeni kako grupa na logičko povrzani aktivnosti i odluki potrebni za upravuvanje so resursite i ostvaruvanje na osnovnite celi i funkcii. Nivnata identifikacija e napravena trgnuvajći od životniot ciklus na osnovnite resursi prstaveni niz četiri fazi i toa: planiranje, obezbeduvanje, održuvanje i nivno koristenje. Alarmanten beše faktot deka za poedini resursi ne postoi definiran proces na planiranje i deka toa se raboti koi se vršat "AD HOCK" po baranje na povisoki organi ili se utvrdeni vo planskite dokumenti, koe beše premnogu generalno za edna vakva analiza.

Od analizata proizlegoa i zaednički vidovi na procesi i toa: pribiranje i obrabotka na podatoci, izgotvuvanje na analizi i informacii i vodenje na arhiva, biblioteka i dokumentacija.

Isto taka gi utvrdivme osnovnite klasi na podatoci koi pretstavuvaat grupa na logičko povrzani informacii, kako potrebni resursi vo odvivanjeto na rabotite i zadačite vo organite na upravata. Definiranjeto na klasite na podatocite e izvršeno na osnova na kategorizacijata na podatocite na viďovi (inventarni, dinamički, planski i istoriski) povrzani so sekoja faza od životniot ciklus na resursite. Pri ovaa analiza utvrdeno e deka vo organite ima zaednički klasi na podatoci koi se obrabotuvaat i koristat. Toa se: gragjani, organizacii na združen trud, teritorijalni edinici i adresi, statistički podatoci, planski indikatori i normativni akti. Koga gi analiziravme postojnite aplikacii koi se koristat vo redovnoto rabotenje na organite na upravata proizleze deka malku e napraveno na poleto na avtomatizacija. Toa se pred se aplikacii svrzani so operativnoto rabotenje na poedini organi a ne značitelno se kaj plansko-rakovodnite i analitičkite procesi.

So cel točno da se sogledaat korelaciite momegju:

- organizacionata struktura i klasite na podatoci koi gi koristi i sozdava taa struktura,
- organizacionata struktura i glavnite procesi za koi odlučuva, gi izvršuva ili učestvuva vo nivnoto izvršuvanje , i
- glavnite procesi i klasite na podatoci koi tie procesi gi sozdavaat ili samo gi koristat,

iskoristena e metodata na grafičko prikažuvanje so koristenje na matrici, pri što najinteresni rezultati se dobieni so analizata na matricata na relaciite na klasite na podatoci so delovnite procesi i povrzanosta na delovnite procesi so organizacionata struktura od kade se gleda deka za rabotnite procesi vo organite odlučuvaat i odgovaraat rakovodnite strukturi, dodeka izvršuvanjeto e vo delokrugot na sektorite i odelenjata.

So obedinuvanje na poedini matrici na institucii vo zaednički na nivo na Grad, dobieni se osnovnite parametri na zaedništvoto na Informacioniot sistem koi se sostojat pred se vo zaednički bazi na podatoci, standardizacija na osnovnite rabotni procesi, avtomatska razmena na službenite informacii i zaedništvo vo planiranjeto i edinstvo vo koristenjeto na informacioniot sistem i školuvanje na potrebniot kadar.

IV FAZA - VERIFIKACIJA NA REZULTATITE OD ANALIZATA

Verifikacijata e izvršena od samite starešini preku specijalno organizirani poedinečni konsultacii. Konsultaciite grižlivo gi podgotvivme so točno utvrdeno scenario a se podgotvija prezentacii na rezultatite na analizata na postojnoto rabotenje, gledano od aspekt na soodvetniot starešina. Konsultaciite imaa dvojna cel:

- prvo - da se verificiraat rezultatite od analizata,
- vtoro - da se sogledaat problemite vo rabotenjeto, kritičnite faktori, da se doznae što očekuva starešinata od idniot Informacionen sistem, na koi punktovi od negovoto rabotenje očekuva direktna pomoš od Informacioniot sistem i da se obezbedi Informacioniot sistem da dade podrška vo procesot na planiranjeto, rakovodenjeto i odlučuvanjeto.

Analizirajki gi beleškite od izvršenite konsultacii napravena e matrica na problemi vo koja site iskažani problemi se kategoriziraa vo pet osnovni kategorii, i toa: organizacioni, problemi pri planiranjeto, problemi kako posledica od kontrola i sledenje na rezultatite od rabotenjeto, operativni i problemi što se posledica od nadvorešni faktori.

Od analizata na problemite proizleze deka organizacionite problemi i problemite koi gi predizvikuvaat nadvorešnite faktori nemožat

da se rešat so avtomatizacija, dodeka, dobra koncepcija na Informacioniot sistem može da gi nadmine problemite od opreativnoto rabotenje koi se posledica na lošoto planiranje i nesoodvetnata kontrola.

Dobienite rezultati bea iznenaduvački za site členovi na timot i za samite starešini. Se potvrdi deka poradi preoptovarenosta so operativno rabotenje našite starešini go zanemaruvaat planiranjeto i kontrolata na kvalitetot na rabotenjeto, zaključok koj bara brza i konkretna akcija.

Konsultaciite so starešinite gi dadoa očekuvanite efekti. Celosno e izvršena verifikacija na rezultatite od analizata i voedno starešinite pokažaa celosna gotovnost da razgovaraat za problemite so koi se soočuvaat i za efektire što gi očekuvaat od idniot Informacionen sistem.

V FAZA - SINTEZA NA REZULTATITE

Sintezata na rezultatite dobieni od analizata na postojnoto rabotenje i vgraduvanje na sugestiite i preporakite dobieni od starešinite po odnos na efektite što gi očekuvaat od idniot Informacionen sistem se vgradeni pri utvrduvanjeto na osnovnata stuktura na integralniot Informacionen sistem. Vo toj kontekst se utvrdeni:

- zaedničkite bazi na podatoci, i

- osnovnite informacioni sistemi što go sočinuvaat interglarniot informacionen sistem.

Sekoj od utvrdenite osnovni informacioni sistemi e razložen na informacioni podsistemi, a sekoj podsistem na konkretni proekti.

Zaedništvoto pomegju definiranite informacioni sistemi se ostvaruva preku: koristenje na zaedničkite bazi na podatoci, sozdavanje na edinstven sistem za ažuriranje na podatocite, sozdavanje na tehnički i organizacioni uslovi za brza i efikasna razmena na informacii, povrzuvanje na avtomatiziranite so neavtomatiziranite evidencii i primena na edinstveni standardi za priprema, obrabotka i prezentiranje na podatocite. Edinstvoto na integralniot Informacionen sistem će se ostvaruva preku distribuirana obrabotka na podatoci pri što, vnesot na podatoci, osnovnite obrabotki i najmasovnite koristenja će se izvršuvaat vo organite koi tie podatoci gi sozdavaat.

Inforamcioniot sistem na grad Skopje vo prva faza će go sočinuvaat šest zaednički bazi na podatoci i sedum nezavisni informacioni sistemi.

Osnovno načelo vo realizacijata na zacrtaniot koncept e da se obezbedi paralelen razvoj na site informacioni sistemi.

Sekoj Informacionen podsistem e razložen na EOP proekti. Pri definiranjeto na prioritetot na proektite se poagjaše od poveće kriteriumi kako što se: objektivna ocena na prioritet vrednuvan po poveće parametri (potencijalni efekti, vlijanie vrz postojnata organizacija, šansi za uspeh i drugo), ocena na prioritet baziran na tehnologija na odvivanjeto na delovnite procesi, i direktni sugestii od starešinite.

Ocenkata na prioritetot vrednuvan po prviot kriterium beše najsubjektivna imajći gi vo predvid učesnicite vo Studijskiot tim i nivnoto tekovno rabotenje. Vo odnos na zaedničkite bazi na podatoci ovoj kriterium beše objektiven. Kako rezultat na ova vrednuvanje bea izgotveni prioritetni listi na proektite i na zaedničkite bazi na podatoci pri što se istakna načeloto da se ovozmoži paralelen razvoj na poedinečnite informacioni sistemi vo soglasnost so objektivnite možnosti, a komponentite na zaedništvo dobija najvisok prioritet.

Trgnuvajći od utvrdenata koncepcija potrebata od zaedništvo i zacrtanite prioriteti, se utvrdi osnovnata konfiguracija na potrebnata kompjuterska oprema što treba da pretstavuva tehnička baza na integralniot Informacioniot sistem na Gradot. Pri toa se koristeni važečkite standardi na toa pole kako i informaciite za možnostite na soveremenata tehnologija što se nudi na našiot pazar.

Za realizacija na predloženata koncepcija potrebnata oprema e so slednite karakteristiki: centralen sistem koj će obezbeduva neprečeno funkcioniranje na informacionite sistemi i razmena na službenite informacii, mreža od terminali za direktna avtomatizacija na rabotnite procesi, dovolno golem magneten medijum koj ovozmožuva čuvanje na golem broj na podatoci, komunikaciska oprema za direkten pristap do podatocite, brzo pečatenje na izvešai, avtomatsko izdavanje na dokumenti na šalterite, obezbeduvanje na sigurnost i zaštita na podatocite i možnost za čuvanje na istoriski podatoci i dokumentacija na soodveten magneten medium. Obezbeduvanjeto na distribuiranata oprema će se vrši zapazuvajći go principot na objektivna operacionalizacija t.e. će se imaat vo predvid efektite što se očekuvaat i materijalnite možnosti koi organite gi imaat za obezbeduvanje na oprema.

VI FAZA - IZGOTVUVANJE NA ZAVRŠEN IZVEŠTAJ, VERIFIKACIJA I USVOJUVANJE

Osnovni principi vo izgotvuvanjeto na završnite dokumenti na Studijata bea: tie da se podgotvuvaat sukcesivno so odvivanje na Studijata i vo nivno izgotvuvanje ramnopravno da bidat vključeni site členovi na timot.

Od Studijata proizlegoa dva dokumenta završen izveštaj i dokumentacija. So ogled deka osnovnite konturi na završniot izveštaj bea odnapred utvrdeni, poodelni temi bea rasporedeni pomegju členovite na timot, a recengenski odbor sostaven od členovi na Studiskiot tim, završniot izveštaj go oformi vo logička celina.

Za oblikuvanje na dokumentacijata na Studijata koristeni se specijalni obrasci utvrdeni so BSP metodologijata i pogodnostite na programskiot produkt za obrabotka na tekst "SCRIPT". Nekoi od ovie obrasci se priloženi vo ovoj referat.

Isto taka beše izgotveno i rezime od izveštajot vo koe na popularen način se iskažani rezultatite od Studijata.

Verifikacijata na Studijata e izvršena od strana na mentorot na Studijata i starešinite na organite, pri što bea prifateni i javno verificirani rezultatite od Stidijata a voedno beše potvrden i planot na aktivnosti za realizacija na Informacioniot sistem, so cel predloženata koncepcija etapno da se realizira.

Utvrdenata koncepcija i nejzinata realizacija ja usvoija Izvšniot sovet na Sobranieto na Grad Skopje i delegatite na Sobranieto na gradot i opštinite od područjeto na Gradot, pri što se istakna potrebata od celosno angažiranje za obezbeduvanje na potrebnite preduslovi za realizacija na utvrdenata koncepcija kako organizacioni taka i tehnički, kadrovski i materijalni.

ZAKLUČNI SOGLEDUVANJA

Ako se pojde od toa deka od Studijata se očekuvaše pred se:

- utvrduvanje na osnovnata koncepcija i struktura na Informacioniot sistem, bazirana na analizata na delovnite procesi i protokot na delovnite informacii,

- obezbeduvanje na zaedništvo vo planiranjeto na Informacioniot sistem i edinstvo vo negovata realizacija,

- obezbeduvanje na aktivna podrška na predloženata koncepcija od strana na rakovodnite strukturi i zajaknuvanje na nivnata doverba vo opravdanosta na planiranite investicii, i

- maksimalno iskoristuvanje na iskustva na rakovoditelite i stručnite sorabotnici so nivno aktivno vklučuvanje vo realizacijata na studijata,

togaš kako iskustvo vo koristenje na BSP metodologijata za planiranje na informacionite sistemi, može da se zaključi slednoto:

1. Primenata na BSP metodologijata vo procesot na planiranje na Informacioniot sistem se pokaža kako efikasen metod koj dokolku dosledno se primeni dava dobri rezultati. Istoto se potvrdi i vo našata Studija koja gi potvrdi očekuvanjata za kvalitetno planiranje na Informacioniot sistem.

2. Bidejći planiranjeto na Informacioniot sistem go vršat krajnite korisnici će gi branat osnovnite postapki na predložената koncepcija da ukažuvaat na očekuvanite efekti i da se zalagaat za nejzina dosledna realizacija.

3. Metodologijata gi obedinuva parcijalnite interesi na oddelni službi vo edinstvena koncepcija na Informacioniot sistem koja obezbeduva zaedništvo vo planiranjeto i edinstvo vo realizacijata.

4. So aktivno vklučuvanje na starešinite vo fazata na verifikacijata na rezultatite od analizata i vgraduvanje na nivnite sugestii vo predloženata koncepcija na Infromacioniot sistem se obvrzuvaat starešinite da bidat ubedeni deka se nudi vistinsko rešenie, koe pokraj avtomatizacija na operativnoto rabotenje direktno će pomogne vo procesot na planiranjeto, rakovodenjeto i odlučuvanjeto.

5. Intenzivnosta vo rabotenjeto kako i širokiot raspon na problemite što se analiziraat baraa disciplina vo rabotenjeto, maksimalna standardizacija na aktivnostite i sposobnost vnimanieto da se nasoči kon osnovnite problemi, odnosno da se vrši selekcija po prioritet.

Kako kraj, možeme dá istakneme deka napravena e Studija, utvrdena e koncepcija, definirani se prioriteti, no ako toa što sme go isplanirale vednaš ne go realizirame togaš gubi od značenje i celata Studija.

hsp04*2

****** BSP STUDIJA ZA GZ GRIS ******

ZAEDNIČKA MATRICA : P R O C E S I / O R G A N I Z A C I S K A S T R U K T U R A

ORGANIZACISKA STRUKTURA ==> / PROCESI	ZAEDNICKI	GZ ORIZ	UP PRIHODI	GEOD. UPRAVA	ZPRG	SIZ ZDRAVSTVO	SIZ VRABOTUVANJE
	a b c d e	f g h i	j k l m n o	p q r s t u v	w x y z	1 2 3 4 5 6 7 8 9	? ! ¬ \ / # !) - - -
1. Planiranje AOP proekti	- @ - - -	X X - -	- - - - - -	- - - - - - -	- - - -	- - - - - - - - -	- - - - - - - - - - - -
2. Realizac. AOP proekti	- @ - - -	/ X - -	- - - - - -	- - - - - - -	- - - -	- - - - - - - - -	- - - - - - - - - - - -
3. Voden. zaednicki banki p.	- @ - - -	/ X - -	- - - - - -	- - - - - - -	- - - -	- - - - - - - - -	- - - - - - - - - - - -
4. Org. seminari i konsult.	- @ - - -	X X - -	- - - - - -	- - - - - - -	- - - -	- - - - - - - - -	- - - - - - - - - - - -
5. Utvrduvanje na prihodi	- @ - / -	- - - -	X - - / - -	- - - - - - -	- - - -	- - - - - - - - -	- - - - - - - - - - - -
6. Vodenje danocno knigovod.	- @ - @ -	- - - -	/ X / - - -	- - - - - - -	- - - -	- - - - - - - - -	- - - - - - - - - - - -
7. Realizacija - NAPLATA	- @ - @ -	- - - -	X / X - - -	- - - - - - -	- - - -	- - - - - - - - -	- - - - - - - - - - - -
8. Odrzuva. premer,katastar	- @ @ @ -	- - - -	- - - - - -	X X - - X - -	- - - -	- - - - - - - - -	- - - - - - - - - - - -
9. Vodenje evid. korisnici	- - - @ -	- - - -	- - - - - -	X X - - - - -	- - - -	- - - - - - - - -	- - - - - - - - - - - -
10. Izdavanjje na dokumenti	- @ - @ -	- - - -	- - - - - -	X X X / - - -	- - - -	- - - - - - - - -	- - - - - - - - - - - -
11. Izdavanje sluzbeni infor.	- @ X @ -	- - - -	- - - - - -	X / X - - - -	- - - -	- - - - - - - - -	- - - - - - - - - - - -
12. Utvrduv. na metodologija	- @ @ @ -	- - - -	- - - - - -	- - - - - - -	X X - -	- - - - - - - - -	- - - - - - - - - - - -
13. Kontrolna funkcija	- @ @ @ -	- - - -	- - - - - -	- - - - - - -	X X _ _	- - - - - - - - -	- - - - - - - - - - - -
14. Obrabotka na podatoci	- - - @ -	- - - -	- - - - - -	- - - - - - -	X X - -	- - - - - - - - -	- - - - - - - - - - - -
15. Izbor na optimal.varijanta	- @ @ @ -	- - - -	- - - - - -	- - - - - - -	X X - -	- - - - - - - - -	- - - - - - - - - - - -
16.	- - - - -	- - - -	- - - - - -	- - - - - - -	- - - -	- - - - - - - - -	- - - - - - - - - - - -
17.	- - - - -	- - - -	- - - - - -	- - - - - - -	- - - -	- - - - - - - - -	- - - - - - - - - - - -
18. Voden. evid.zdrav.osigur.	- @ @ @ -	- - - -	- - - - - -	- - - - - - -	- - - -	- - - - X - - - -	- - - - - - - - - - - -
19. Retaksacija na recepti	- @ @ @ -	- - - -	- - - - - -	- - - - - - -	- - - -	- - - X - - - - -	- - - - - - - - - - - -
20. Finansiranje	- @ @ @ -	- - - -	- - - - - -	- - - - - - -	- - - -	- @ X X - - - - -	- - - - - - - - - - - -
21. Izgotvuv.planski dokumen.	- @ @ - -	- - - -	- - - - - -	- - - - - - -	- - - -	X X - - - X - / -	- - - - - - - - - - - -
22. Voden.evid. nevrabotni	- / - @ -	- - - -	- - - - - -	- - - - - - -	- - - -	- - - - - - - - -	/ X - - - - - - - - - -
23. Posreduvanje pri vrabot.	- @ - @ -	- - - -	- - - - - -	- - - - - - -	- - - -	- - - - - - - - -	X X - - - - - - - - - -
24. Osposobuv. i prekvalifik.	- @ - @ -	- - - -	- - - - - -	- - - - - - -	- - - -	- - - - - - - - -	X / - - - - - - - - - -
25. Kreditiranje,participac.	- @ - @ -	- - - -	- - - - - -	- - - - - - -	- - - -	- - - - - - - - -	/ / - X X X - - - - - -
26. Smetkovodstvo	- @ - @ -	- - - -	- - - - - -	- - - - - - -	- - - -	- - - - - - - - -	- - - - - X - - - - - -
27. Voden.evid. za vrabotuv.	- / - @ -	- - - -	- - - - - -	- - - - - - -	- - - -	- - - - - - - - -	/ X - X X - - - - - - -
28. Sobiranje na podatoci	- - - @ -	- - - -	X X / - - -	- - - - - - -	X X - -	- - - - - - - - -	- - - - - - - - - - - -
29. Obuka za AOP (korisnici)	- @ - - -	X X - -	- - - - - -	- - - - - - -	- - - -	- - - - - - - - -	- - - - - - - - - - - -
30. Izrab. analizi i inform.	- @ @ @ -	- - - -	- - - X - -	/ / / / / / -	X X - -	- X - - - X - / -	/ / X X - - - - - - - -
31. Vodenje arhiva i dokumen.	- @ @ @ -	X X - -	X X X X - -	X X X X X / -	X X - -	- - - - - X - - -	- - - - - - - - - - - -

legenda : @ - odlicuva   x - izvrsuva   / - ucestvuva

ZAEDNIČKI STRUKTURI: a - Sobranie b - Direktor/sekretar c - Pom.direktor/pom.sekretar d - Načalnik/rakovod.

**** BSP GZ URIS ****

DATUM. 29.08.84

TERMINSKI PLAN : STUDIJA I PLAN ZA REALIZACIJA NA B S P S T U D I J A

RABOTNI AKTIVNOSTI	20.08.	27.08.	03.09.	10.09.	17.09.	24.08.	01.10.	08.10.	15.10.	22.10.	29.10.
1. PODGOTOVKI ZA OTPOČNUVANJE NA STUDIJATA	XXXXX	XXXXX									
2. ŠKOLUVANJE SEMINAR FAO BSP METOD. RADOVLJICA/SLOVENIJA			-XXXX								
3. IDENTIFIKUVANJE I DEFINIRANJE NA RABOTNITE PROCESI I KLASI NA PODATOCI				XXXXX	XXXXX						
4. VERIFIKACIJA NA REZULTATITE PREKU RAZGOVORI/INTERVJUI SO STAREŠINITE I RAKOVODITELITE NA POEDINI ORGANI						XXXXX					
5. SINTEZA NA REZULTATITE I IZRABOTKA NA PREDLOG REŠENIE NA INFORMACIONIOT SISTEM NA SKOPJE							XXXXX				
6. PPIPREMA NA ZAVRŠNIOT IZVEŠTAJ NA STUDIJATA						XXXXX	XXXXX	XXXXX			
7. PREZENTIRANJE NA REZULTATITE OD STUDIJATA									--X--		

ZABELEŠKI :

INFORMACIJSKI SISTEM ZA DELO REPUBLIŠKIH ORGANOV IN ORGANIZACIJ IZKUŠNJE PRI UVAJANJU TER MOŽNOSTI ZA NADALJNJI RAZVOJ

Erih Skočir,
Sekretariat Izvršnega sveta Skupščine SR Slovenije,
Ljubljana, Jugoslavija

UDK: 681.3.007

Referat daje kratek pregled razvoja, ter povzema izkušnje pri graditvi in uvajanju informacijskih sistemov s poudarkom na informacijskem sistemu za delo republiških upravnih organov in organizacij. Navedene so tudi glavne značilnosti informacijskega sistema (enotna metodologija dela, enotni standardi ipd.) ter prednosti in težave, ki nastajajo pri njegovem uvajanju. Na koncu je podana tudi možnost za nadaljnji razvoj s ciljem še večjega uveljavljanja procesa družbenega sistema informiranja in razvoja informacijskih sistemov za podporo odločanju.

INFORMATION SYSTEM FOR WORK IN OFFICIAL REPUBLIC ORGANIZATIONS EXPERIENCES GAINED AT INTRODUCING SUCH SYSTEM AND POSSIBILITIES FOR FURTHER DEVELOPMENT

The paper gives a short survey of development and presents experiences gained at introducing information systems with a special stress upon the information system for work in official republican organizations. Main characteristics of the information system (uniform methodology of work, uniform standards, etc.) as well as advantages and difficulties occurring during and with its introduction are stated. At the end, also a possibility for further development with a goal to bring forward the process of social information system and to expand information systems for computer aided decision making is implied.

1. UVOD

Temelji o razvoju družbenega sistema informiranja, zastavljeni v zakonskih aktih (1, 2), zavezujejo ter omogočajo delavcem, drugim delovnim ljudem in občanom, da druge seznanjajo in da so sami seznanjeni z vsemi pojavi in odnosi, ki so pomembni za njihovo življenje, delo in odločanje. Ob že obstoječih metodah in načinih dela pa se z uporabo sodobne tehnologije in tehnične opreme odpirajo nove razsežnosti.

Kljub vsem prizadevanjem, razen nekaj posameznih poskusov, se je šele letos pokazala možnost za uresničitev idej. S sprejemom samoupravnega sporazuma o sodelovanju in izvajanju skupnih nalog pri načrtovanju, vzpostavljanju in delovanju informacijskih sistemov za podporo odločanju so dane možnosti za enotne in usklajene akcije.

Osnovne skupne značilnosti razvoja vseh informacijskih sistemov za podporo odločanju so:

- medsebojno usklajena metodologija zajemanja, obdelave, shranjevanja in izkazovanja podatkov in inforamcij;
- medsebojno usklajena standardizacija nastajanja, zbiranja, obdelave, shranjevanja in izkazovanja podatkov in informacij;
- usklajena strojna in programska oprema in njeno vzdrževanje;
- enotne oblike in postopki arhiviranja dokumentov (ne glede na medij);
- enotne oblike in metode usposabljanja in izobraževanja;
- enotni postopki zaščite in varovanje podatkov in informacij (ne glede na medij).

Glavni cilji teh usklajenih akcij pa so:

- boljše obveščanje in odločanje;
- povečana kakovost strokovno - političnega dela;
- racionalizacija administrativno - strokovnega dela.

Podlaga informacijskih sistemov za podporo odločanja so:

- baze podatkov informacijskih služb skupnega pomena za republiko
- skupne in večnamenske baze podatkov drugih informacijskih služb

Eden od razvitih informacijskih sistemov, ki pokriva samo del nalog pri razvoju informacijskih sistemov za podporo odločanju, je informacijski sistem za delo republiških organov, vzpostavljen v Sekretariatu Izvršnega sveta Skupščine SR Slovenije.

2. INFORMACIJSKI SISTEM ZA DELO REPUBLIŠKIH UPRAVNIH ORGANOV IN ORGANIZACIJ

2.1. Dosedanji razvoj informacijskega sistema

Že leta 1978, pa tudi prej, so se porajala razmišljanja o posodobitvi poslovanja Izvršnega sveta Skupščine SR Slovenije. Vendarle projekti niso bili realizirani vse do leta 1979. Septembra je bil v Sekretariatu Izvršnega sveta Skupščine SR Slovenije ustanovljen INDOK center kot nosilec razvoja posodobitve poslovanja dela Izvršnega sveta. Z Republiškim sekretariatom za notranje zadeve, ki je od takrat tudi nosilec tehnične osnove, je bilo dogovorjeno, da pomaga tudi pri uvajanju in izdelavi programov za razvoj informacijskih sistemov. Tako je že 1980. leta zaživel informacijski sistem za delo Izvršnega sveta, ki je kasneje prerasel v informacijski sistem za delo rapubliških organov in organizacij.

V začetku so se nekatera dela podvajala, nismo bili vedno na tekočem, imeli smo 'otroške bolezni'. Z razvojem informacijskih sistemov, metodologije, standardizacije smo se tudi vsi delavci delovne skupnosti veliko naučili. Zaradi novega načina dela smo morali nekajkrat spreminjati naše samoupravne akte (sistemizacijo, pravilnik o nagrajevanju delavcev ipd.).

Po prvih uspehih pri uvajanju informacijskega sistema za delo Izvršnega sveta Skupščine SR Slovenije smo v INDOK sektorju ugotovili, da bi se delo lahko še izboljšalo, če bi sodobna tehnična sredstva vključili neposredno tja, kjer informacije nastajajo ali kjer jih potrebujejo. Zato smo začeli z distribuirano obdelavo. S tem smo sprožili tudi postopek za ukinitev INDOK sektorja, ker ni bila več potrebna ena služba, ki bi skrbela za vnos, pripravo in obdelavo podatkov. Da smo lahko vključili terminale v obstoječe delovne procese, smo morali spoznati vsak delček delovnega procesa ter pri zamenjavi delovnih postopkov prepričati delavce, da je novinačin dela boljši. To pa smo lahko dosegli, le tako, da smo delo olajšali, čas obdelave skrajšali, zagotovili zamenljivost delavcev, omogočili lažje popravljanje dokumentov ipd. Na ta način smo dosegli hitrejši vnos dokumentov v bazo podatkov z manj truda. Administrativno delo se je zmanjšalo od 10% do 60% (ni več treba trikrat ali večkrat tipkati isti dokument - zapisnik ali podobno). Hkrati pa smo dokumente shranili v bazo podatkov ter tako vsem uporabnikom, ki imajo v bližini tehnično opremo, omogočili dostop do želenih informacij.

Ob vzpostavljanju informacijskih sistemov smo sproti dopolnjevali tehnično opremo (terminali, tiskalniki, mikrofilmski čitalniki idr.), ki je že skoraj v vseh organizacijskih enotah Sekretariata Izvršnega sveta Skupščine SR Slovenije.

2.2. Osnovne značilnosti informacijskega sistema

Skupno vsem informacijskim sistemom v Sekretariatu Izvršnega sveta Skupščine SR Slovenije je, da je obdelava distribuirana (dokumentacijska, računalniška in mikrofilmska).

Računalnik je nosilec kratke primarne informacije, originalna informacija pa je posneta na mikrofilmu. Preko sistema COM je mogoče iz skupne baze podatkov dobiti le-te na mikrofilmu, urejene po merilih, ki jih postavljajo uporabniki.

Vsak uporabnik se mora naučiti samo nekaj osnovnih operacij v informacijskem sistemu in se že lahko aktivno vključi v delo.

Uporabnik nima nobenega opravka s programsko opremo ali vzdrževanjem tehnične opreme.

Vsi udeleženci v sistemu uporabljajo enotno standardizacijo, kar omogoča vsebinsko spremljanje podatkov z možnostjo analize in spremljanja realizacije ne glede na fazo obdelave določenega gradiva.

Vsi postopki dela so tudi standardizirani, tako da je možna zamenljivost delavcev, neglede na to v kateri organizacijski enoti delavec dela (če je le seznanjen z osnovnimi postopki). Razen tega se pri pripravi in vnosu podatkov več ne uporabljajo nikakršni obrazci, ker so vse oblike dokumentov standardizirane in shranjene v računalniku.

Za vse faze obdelav dokumentov v informacijskem sistemu so izdelana tudi navodila, ki omogočajo, da se vsak lahko uči sam.

Programska oprema trenutno omogoča delo v on-line načinu pri vnosu podatkov in iskanju informacij, kar pa se bo verjetno z razvojem osebnih računalnikov v prihodnosti nekoliko spremenilo.

2.3. Programska oprema

Za uspešen razvoj informacijskega sistema za delo republiških upravnih organov je potrebna ustrezna programska podpora. Ker so v splošnem naloge razdeljene v dve skupini (vnos podatkov in iskanje podatkov iz skupne baze) je tudi programska podpora temu prirejena.

V informacijskem sistemu za spremljanje dela republiških organov zato uporabljamo dva programska paketa:

a) ATMS (Advanced Text Management System),
   ki omogoča distribuiran vnos podatkov, njihovo editiranje, formatiziranje in pripravo za kasnejši prenos v skupno bazo podatkov;

b) STAIRS (STorage And Information Retrieval System),
   ki se uporablja za hitro in učinkovito iskanje podatkov in dokumentov preko terminala.

Osnovni namen programskega paketa ATMS je, priprava dokumentov v takšni obliki, da so primerni za polnjenje skupne baze podatkov. Paket omogoča tudi vsa druga dela v zvezi z urejevanjem besedil (formatiziranje besedila, avtomatsko oštevilčevanje strani, usrediščenje besedila, kopiranje, premikanje besedila, izpisovanje besedila na kakršnikoli format idr.). Ta njegove lastnosti so bile še posebej koristne, ko smo prehajali na dristribuirano obdelavo dokumentov in ko smo racionalizirali celotno administrativno poslovanje.

Dokumenti, shranjeni v skupni bazi podatkov, se s pomočjo programskega paketa STAIRS lahko iščejo:

a) na osnovi posameznih operandov (besed, pojmov, številk, korenov besed ipd.) in njihovih medsebojnih logičnih odnosov

b) na osnovi konstant, prilagojenih matematičnim relacijam

c) na osnovi pogostosti posameznih operandov v določenem dokumentu

Na osnovi znanih podatkov in naštetih načinov iskanja se iz baze podatkov izloči določeno število dokumentov, ki ustreza vsem danim merilom. Izbrano število dokumentov lahko prikazujemo na zaslonu, razvrščamo po formatiziranih poljih, izpisujemo na papir itd. STAIRS tudi omogoča shranjevanje zaporedja zahtevkov in njihovo izvajanje, hiter preskok v druge baze podatkov, samoizobraževanje z datoteko HELPMSG, prikaz in izpis določenih ali vseh zahtevkov z njihovimi razultati itd.

2.4. Zavarovanje in zaščita podatkov

Posebna pozornost pri razvoju in uvajanju informacijskega sistema za delo republiških organov je posvečena zavarovanju in zaščiti podatkov in dokumentov v vseh fazah obdelave. Ker je bila večina postopkov v zvezi z zavarovanjem in zaščito podatkov pri dokumentacijski obdelavi že določena, smo morali v samoupravne splošne akte vključiti določila, ki se nanašajo na zavarovanje in zaščito dokumentov pri računalniški in mikrofilmski obdelavi. Računalniški obdelavi je posvečena posebna pozornost posebej pri STAIRS-u, ker se dokumenti v ATMS-u hranijo samo toliko časa, dokler niso preneseni v skupno bazo podatkov.

Vse vrste omenjene zaščite so podrobno opisane v literaturi (10).

2.5. Kadrovanje in izobraževanje

Pri uvajanju informacijskih sistemov so največje težave zaradi pomanjkanja ustreznih strokovnjakov ter nizke ravni računalniške pismenosti delavcev.

Kadrovska politika mora zagotoviti strokovnjake za vzdrževanje skupnih baz informacijskih sistemov ter uvajanje informacijskih sistemov v posamezne upravne organe in organizacije. Torej, mora biti enotna in usklajena.

Vzporedno mora potekati izobraževanje vseh delavcev. Čeprav bi vsi morali, vsaj informativno, poznati vse faze obdelave dokumentov, so v glavnem procesi izobraževanja usmerjeni na dve glavne smeri:

a) izobraževanje za pripravo, vnos in obdelavo dokumentov (administrativni delavci)

b) izobraževanje za iskanje informacij iz skupne baze podatkov (strokovni delavci)

Veliko pozornosti je treba posvetiti izdelavi navodil za delo in samoučenje. Izobraževanje je stalen proces, ker je od njega največ odvisno, kako se bodo posamezni informacijski sistemi vključili v obstoječe delovne procese. Večinoma je od tega odvisno, ali se bo informacijski sistem sploh uveljavil ali ne.

2.6. Nadaljnji razvoj informacijskega sistema

Če hočemo zagotoviti nadaljnji uspešni razvoj informacijskih sistemov za delo republiških organov in organizacij moramo:

- poenotiti delovne postopke in zagotoviti možnosti za uporabo enotne metodologije pri zajemu, obdelavi, in iskanju podatkov, informacij in dokumentov;
- graditi ustrezno tehnično infrastrukturo (pri tem posebej posvetiti pozornost uporabi novih tehnoloških rešitev);
- zagotavljati razvoj skupne programske opreme;
- razvijati vse oblike izobraževanja delavcev in krepiti načrtno in usklajeno kadrovsko politiko;
- pospeševati druge dejavnosti, ki prispevajo k večji racionalizaciji dela in boljšemu odločanju;
- razvijati sodelovanje z znanstvenimi ustanovami in drugimi organizacijami, ki se ukvarjajo z informatiko.

Tukaj so naštete samo nekatere osnovne smeri delovanja. Za uspešen razvoj pa je potrebno še veliko drugih usklajenih dejavnosti vseh delavcev republiških upravnih organov in organizacij, pa tudi širše družbene skupnosti.

3. SKLEPNE UGOTOVITVE

Razvoj posameznih informacijskih sistemov republiških upravnih organov, pa tudi širše, zahteva načrtnost in usklajenost akcij na vseh področjih družbenega delovanja. Seveda pa morajo biti ta prizadevanja stalna, ker lahko edino s postopnim premagovanjem vseh preteklih in novih problemov racionaliziramo in moderniziramo poslovanje republiških upravnih organov in družbe kot celote, s tem pa tudi zagotovimo boljše obveščanje in odločanje ter novo kvaliteto pri razvoju družbenega sistema informiranja.

LITERATURA:

(1) Zakon o družbenem sistemu informiranja Ur. list SR Slovenije, št. 10/1983

(2) Zakon o temeljih družbenega sistema informiranja in o informacijskem sistemu federacije
Ur. list SFRJ, št. 68/1981

(3) Dušan Verbič: Nekaj značilnosti razvoja informacijskih sistemov republiških organov
Zagreb, Društveni sistemi informiranja 1983

(4) Viktorija Vrezec: Razvoj standardizacije pri informacijskem sistemu za delo republiških organov
Zagreb, Društveni sistemi informiranja 1983

(5) Cene Bavec: Informacijski sistemi za podršku odlučivanju u upravi
Zagreb, Društveni sistemi informiranja 1983

(6) Alenka Malešič: Informacijski sistem za odločanje v Skupščini SR Slovenije
Zagreb, Društveni sistemi informiranja 1984

(7) Skupne usmeritve razvoja informacijskih sistemov republiških organov, organizacij in služb v republiki
Izvršni svet S SRS, Ljubljana, 1983

(8) Ivan Bratko: Intelegentni informacijski sistemi
Fakulteta za elektrotehniko v Ljubljani, 1982

(9) Samoupravni sporazum o sodelovanju in izvajanju skupnih nalog pri načrtovanju vzpostavljanju in delovanju informacijskih sistemov za podporo odločanju
Skupščina SRS, Ljubljana, 1985

(10) Sekretariat IS Skupščine SRS in RSNZ: Računalniška in mikrofilmaska obdelava podatkov v Sekretariatu Izvršnega sveta Skupščine SR Slovenije
Sekretariat IS S SRS, Ljubljana, 1980

PROGRAMSKI PAKET ZA VODENJE EVIDENCE INVESTICIJSKIH PROGRAMOV

Viljan MAHNIČ, Samo ERŽEN, Primož BELTRAM
Univerza Edvarda Kardelja
Fakulteta za elektrotehniko
Ljubljana, Yugoslavia

UDK: 681.3.06

POVZETEK: Poročilo opisuje programski paket za vodenje podatkov o investicijskih programih, ki omogoča avtomatsko generiranje različnih seznamov, poročil in analitičnih tabel s sumarnimi podatki o investicijah v teku. Paket je bil razvit za osebni računalnik Partner na Fakulteti za elektrotehniko v Ljubljani in se uporablja na Institutu za ekonomiko investicij pri Ljubljanski banki.

A PROGRAM PACKAGE FOR THE MANAGEMENT OF INVESTMENT PROGRAM DATA

SUMMARY: We describe a program package which stores and maintains data of investment programs. This package enables the automatic generation of various lists, reports and analitical tables which contain summarized data of investments in progress. It was developed for the Partner personal computer at the Faculty of Electrical Engineering, Ljubljana and is used at the Ljubljana Bank, Institute for the Economics of Investments.

1. Uvod

V času, ko so sredstva za investicije omejena, je nadzor nad investicijami in razporejanjem sredstev zanje še posebej pomemben. Zato so se na Institutu za ekonomiko investicij pri Ljubljanski banki - Združeni banki odločili za računalniško vodenje evidence investicijskih programov, ki jih tam obdelujejo tako za potrebe posameznih temeljnih bank kot tudi na zahtevo investitorjev ali republiških upravnih organov.

V ta namen je bil v sodelovanju s Fakulteto za elektrotehniko iz Ljubljane razvit programski paket, ki omogoča zajem in hranjenje podatkov o posameznih investicijah ter na osnovi teh podatkov avtomatsko generira razna poročila in analitične tabele, ki jih banka potrebuje za svoje delo oziroma jih posreduje ustreznim organom in institucijam. Podatke za te tabele so prej zbirali ročno, kar je bilo izredno zamudno in je predstavljalo za delavce Instituta precejšnjo obremenitev.

Z vpeljavo računalniškega vodenja evidence investicijskih programov se je tako po eni strani bistveno zmanjšal obseg zamudnega ročnega dela, po drugi strani pa so na Institutu za ekonomiko investicij dobili na voljo hitre in kvalitetne informacije, ki omogočajo boljše odločanje o razporejanju sredstev in nadzor nad sredstvi, ki jih banka namenja za kreditiranje posameznih investicij.

2. Splošen opis programskega paketa

Paket je napisan v programskem jeziku PASCAL/MT+ za računalnik PARTNER, sestavljata pa ga dve osnovni skupini programov:

- programi za vzdrževanje podatkovne baze
- programi za izdelavo raznih poročil in analitičnih tabel.

Naloga prve skupine programov je omogočati zajemanje, ažuriranje, brisanje in izpisovanje podatkov, ki jih hranimo o posameznih investicijskih programih. Trenutno hranimo za vsak investicijski program približno 100 najpomembnejših podatkov, ki so organizirani v treh datotekah:

- datoteki splošnih podatkov
- datoteki s podatki o dinamiki izgradnje
- datoteki s podatki o instaliranih zmogljivostih.

Programi iz druge skupine pa so namenjeni za hitro in enostavno izločanje podatkov iz podatkovne baze v obliki raznih poročil, statističnih pregledov in analitičnih tabel. Programski paket omogoča izdelavo naštetih poročil ob upoštevanju naslednjih kriterijev:

- datuma prejema investicijskega programa
- usmerjenosti investicije oziroma področja gospodarstva
- interne projektne klasifikacije.

Možna je poljubna kombinacija naštetih pogojev.

Doslej so ta poročila izdelovali sodelavci Instituta za ekonomiko investicij ročno, kar je bilo po eni strani zelo zamudno, po drugi strani pa je to za njih predstavljalo dodatno obremenitev. Z vpeljavo računalniško vodene evidence sta bili obe navedeni pomanjkljivosti odpravljeni, s tem pa je bila dana tudi možnost hitrejšega in učinkovitejšega spremljanja investicij, ki so v teku.

Poleg omenjenih dveh skupin programov so v paket vključeni še t.i. sistemski programi, katerih delovanje je uporabniku skrito, vendar so nujni za funkcioniranje programskega paketa. To so programi za sortiranje podatkov, iskanje podatkov v bazi, vzdrževanje šifrantov in krmilnih datotek ipd.

Paket je v celoti interaktiven, njegova glavna odlika pa je izredno enostavna uporaba. Dostop do posameznih programov je realiziran preko ustreznih menujev, dialog med uporabnikom in računalnikom pri vpisovanju podatkov pa je zasnovan tako, da računalnik izpisuje zahtevke za vnos, uporabnik pa vnaša potrebne podatke. Vsak vnešen podatek se sproti preveri, računalnik pa v primeru napake izpiše ustrezno diagnozo in zahteva ponoven vnos podatka.

S tem smo dosegli, da lahko uporabljajo paket vsi sodelavci Instituta za ekonomiko investicij, ne glede na računalniško predznanje.

3. Nekatere zanimivejše tehnične rešitve

Od tehničnih rešitev želimo omeniti predvsem uporabo krmilnih datotek, s pomočjo katerih smo dosegli relativno visoko stopnjo neodvisnosti med programi in podatki. Tako smo uvedli posebno krmilno datoteko, ki vsebuje vse pomembnejše informacije o podatkih, ki jih hranimo v podatkovni bazi:

- kje se nahaja posamezen podatek (na kateri datoteki in na katerem mestu v zapisu)
- koliko mest zavzame vsak podatek
- tip podatka oziroma vrste kontrol, ki jih je treba izvršiti pri vnosu oziroma ažuriranju
- besedilo zahtevka za vnos.

S tem je omogočeno enostavno širjenje podatkovne baze z novimi podatki brez izvajanja bistvenih posegov v obstoječih programih, programi za vnos in ažuriranje pa so splošno uporabni in neodvisni od konkretnega nabora podatkov.

Druga tehnična rešitev, ki se nam zdi vredna omembe in ki bistveno prispeva k enostavni uporabi programskega paketa, je postopek verižnja programov. Celoten programski paket poženemo s klicem enega samega (povezovalnega) programa, katerega naloga je pravilno povezovati in krmiliti delovanje vseh ostalih. Ta program izpiše na zaslon glavni menu in nato - v skladu z opcijo, ki jo izbere uporabnik - sproži izvajanje verige ustreznih programov v programskem paketu. Zadnji program v verigi prenese krmiljenje spet na povezovalni program, tako da se na zaslonu - po zaključku vsake izbrane aktivnosti - vedno pojavi glavni menu, na podlagi katerega lahko uporabnik izbira možnosti za nadaljnje delo.

4. Sestava delovne skupine, ki je sodelovala pri realizaciji programskega paketa

Pri realizaciji paketa so sodelovali:

- en sodelavec Fakultete za elektrotehniko (asistent) kot vodja projekta, ki je nosil glavno breme v fazi sistemske analize, izdelal načrt sistema in programske specifikacije ter vodil in nadzoroval potek programiranja
- dva študenta 3. letnika smeri Računalništvo in informatika, ki sta na osnovi programskih specifikacij napisala vse programe
- trije sodelavci Instituta za ekonomiko investicij - uporabniki, ki so bili vključeni predvsem v začetnih in končnih fazah izdelave paketa (analiza, testiranje in uvajanje).

Tako je bil ta projekt zanimiv tudi z vidika vključevanja študentov v praktično delo na razvoju programskih paketov. Izkazalo se je, da lahko študentje ob primerni organizaciji dela uspešno izvršujejo zastavljene naloge, s tem pa pridobivajo izkušnje za svoje delo v praksi po končanem študiju.

5. Zaključek

Programski paket za vodenje evidence investicijskih programov omogoča trenutno izdelavo preko 10 različnih seznamov, poročil oziroma analitičnih tabel, katerih oblika je vnaprej določena. V nadaljnjem razvoju paketa pa nameravamo posvetiti večjo pozornost izdelavi avtomatskega generatorja poročil, ki bo omogočal takojšnjo izdelavo poljubnih poročil, s tem pa bistevno razširil možnosti za kvalitetno spremljanje podatkov o investicijskih programih.

Druga predvidena razširitev programskega paketa pa zajema področje finančnih projekcij, v okviru katerega bomo skušali (izhajajoč iz podatkov v podatkovni bazi) razviti programe za ocenjevanje različnih pokazateljev učinkovitosti posameznih investicij.

IZDELAVA PAKETA "MENIČNO POSLOVANJE" ZA LJUBLJANSKO BANKO GOSPODARSKO BANKO

R. Kolar, M. Toni, Institut Jožef Stefan, Ljubljana, Jugoslavija

I. Lajovic, F. Žerdin, Ljubljanska banka Združena banka, Ljubljana, Jugoslavija

UDK: 681.3.007

Referat opisuje paket "Menično poslovanje" za računalniško podprto obdelavo nekaterih bančnih poslov z vrednostnimi papirji. Referat je razdeljen na tri poglavja. V prvem je nakazana problematika, ki jo paket obdeluje, v drugem je opisan načrt računalniške obdelave in v zadnjem kratek kronološki pregled nastanka paketa. Paket je napisan v programskem jeziku COBOL in uporablja sistem za upravljanje podatkovnih baz IDA. Paket trenutno teče na računalniku DELTA 4780.

THE "BILLS OF EXCHANGE" SOFTWARE PACKAGE

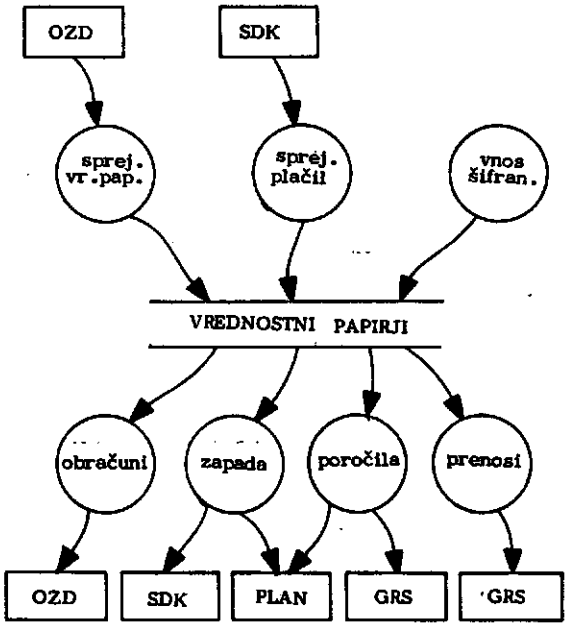
This paper highlights the evaluation of the "Bills of exchange" bank software package. The first chapter describes the objectives, the second the design of the software package and the third the development of the package. The package is writen in the COBOL programming language and uses the IDA database. It is implemented on a DELTA 4780 computer.

1. KRATEK OPIS PROBLEMA

V paketu "Menično poslovanje" so zajeta naslednja področja poslovanja z menicami, garancijami in deviznimi krediti v sektorju združevanja dela in sredstev OZD, v nadaljnjem besedilu "vrednostni papirji" :

- eskont menic, izdanih po zakonu o zavarovanju plačil med UDS z rokom zapadlosti do 90 dni
- eskont menic za prodajo opreme na kredit na domačem trgu
- izdaja garancij G-1 in G-1i za zavarovanje plačila obveznosti iz naslova prometa blaga in opravljanja storitev
- avaliranje menic do 90 dni
- avaliranje menic iz naslova investicij
- storitvene garancije
- devizne garancije
- devizni krediti

Grob pretok podatkov v paketu prikazuje slika 1.



Sl. 1. Pretok podatkov v paketu

Sprejem vrednostnih papirjev zajema ročno preverjanje, če so dokumenti neoporečni, ter vnos v računalnik.

Sprejem plačil pomeni vnos virmanov, s katerimi OZD poravnavajo svoje obveznosti do temeljne banke.

Vnos šifrantov zagotavlja ažurnost splošnih podatkov, ki se občasno spreminjajo (knjigovodski računi, obrestne mere,...).

Zapadanje združuje vse akcije, ki nastanejo, ko vrednostnemu papirju poteče rok veljavnosti, s vsemi knjigovodskimi spremembami in dopisi organizacijam združenega dela.

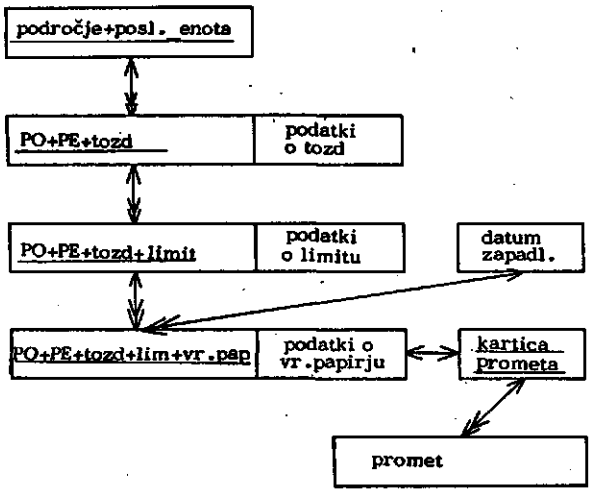
Obračuni združujejo obračune obresti ob odkupu menic, provizije za avale, garancije in kredite, ter zamudne obresti za nepravočasno poravnane obveznosti OZD do temeljne banke.

Knjigovodski prenosi združujejo vse spremembe na knjigovodskih računih in predstavljajo obenem vmesnik med paketom "Menično poslovanje" in obdelavo prometa na knjigovodskih računih v paketih "Saldakonti" in "Glavna knjiga". Paketa "Saldakonti" in "Glavna knjiga" so izdelali delavci Sektorja za računalniški informacijski sistem Gospodarske banke v Ljubljani.

Poročila omogočajo globalen pregled nad podatki, kot na primer:

- *rokovnik zapadlih vrednostnih papirjev*
- *stanje vrednostnih papirjev v določenem obdobju*
- *stanje obračunane in plačane provizije*

2. NAČRT RAČUNALNIŠKE OBDELAVE



Sl.2 Entitetni model

Entitetni model je zasnovan in realiziran tako, da se podatki posameznih področij med seboj ne prepletajo in obenem omogoča, da se tudi podatki več poslovnih enot za isto področje ne prepletajo. To pomeni, da je mogoče izdelati poročilo, ne da bi prišlo do odvečnih čitanj po fizičnih datotekah.

Moduli, ki so navedeni na sliki 3, ustrezajo funkcionalno aktivnostim, ki so opisane na sliki 1. Razdelitev na interaktivne in paketne obdelave nam da :

interaktivne obdelave :

- *zajem in popravljanje podatkov o vrednostnih papirjih*
- *vnos prometa*
- *vnos šifrantov*

paketne obdelave :

- *obračuni*
- *zapadanje*
- *knjigovodski prenosi*
- *poročila.*

Paket je zasnovan na načelu, da uporabniki sami odgovarjajo za pravilnost podatkov.

3. Izdelava paketa

Pri realizaciji projekta je sodelovalo 12 delavcev in sicer so paket skupaj izdelali delavci Odseka za uporabno matematiko Instituta Jožef Stefan in delavci Sektorja za računalniški informacijski sistem Gospodarske banke Ljubljana. Sočasno je na projektu delalo od 4 do 8 delavcev. S strani končnih uporabnikov je sodelovalo 6 delavcev. Izdelava paketa je potekala v grobem v naslednjih fazah :

- *preeliminarna analiza za vsa področja*
- *informacijska analiza za vsa področja*
- *načrt novega sistema za vsa področja*
- *implementacija po posameznih področjih*
  - *programiranje*
  - *testiranje*
  - *uvajanje uporabnikov*
  - *redna produkcija*

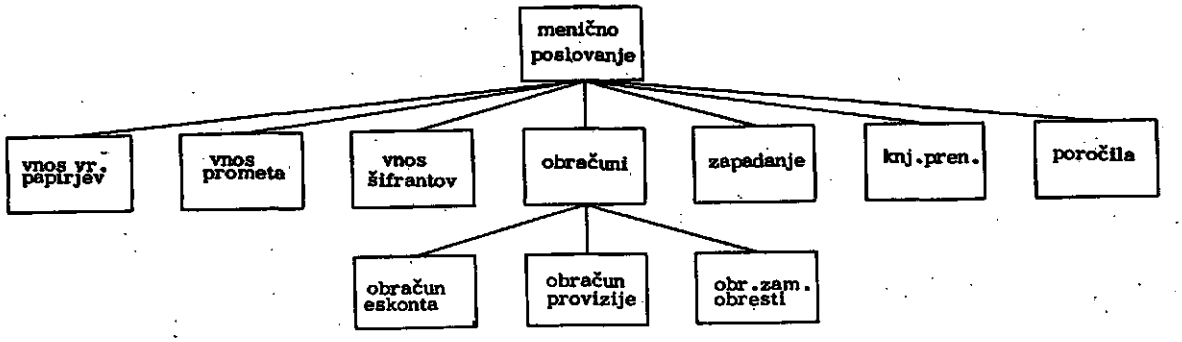
Razvoj paketa je potekal na računalniku ISKRA DELTA 4780. Uporabili smo sistem za upravljanje podatkovnih baz TOTAL ter programski jezik COBOL. Za implementacijo posameznega področja smo potrebovali v povprečju tri mesece koledarskega časa. Paket je začel delati marca 1983 za področje "Eskont menic do 90 dni".

Za testiranje in uvajanje uporabnikov je na voljo testna podatkovna baza, tako da so imeli uporabniki pred začetkom dela nekaj dnevno šolanje, ki so ga v celoti organizirali delavci Gospodarske banke.

Letni obseg podatkov, ki jih uporabniki vnesejo v računalnik, se giblje okrog 200.000 vrednostnih papirjev.

V letu 1985 smo celoten paket predelali na sistem za upravljanje podatkovnih baz IDA, ki ga je izdelala Iskra Delta. Z uporabo IDA baze smo se rešili problemov, ki so nastajali ob nasilni prekinitvi dela računalnika.

Trenutno uporabljajo paket Gospodarska banka Ljubljana in poslovni enoti Kamnik in Kočevje.



Sl.2. Grob strukturni načrt računalniške obdelave

PROGRAMSKI PAKET APP-1

M. VINTAR, VIŠJA UPRAVNA ŠOLA, LJUBLJANA

JUGOSLAVIJA

UDK: 681.3.06

Programski paket APP-1 sodi na področje avtomatizacije pisarniškega poslovanja, ki je doslej še razmeroma slabo obdelano. Omogoča avtomatizacijo evidenc o celotnem dokumentarnem gradivu neke organizacije. Pod dokumentarnim gradivom razumemo vso poslovno korespondenco in vsa druga poslovna gradiva, ki nastajajo v organizaciji ali prihajajo vanjo od zunaj, projektno dokumentacijo, sejna gradiva ter evidenco sklepov. Sistem temelji na vsebinskem razvrščanju gradiv na osnovi klasifikacijskega načrta, kar nam omogoča enostavno iskanje shranjenih informacij.

THE PROGRAMMING PACKAGE APP-1

The programming package APP-1 is designed to cover the part of office automation which has not been thoroughly researched so far. It provides automation of files on business documentation in a firm. Business documentation includes all business corespondence records and other documents generated in the firm or coming from the outside, project documentation, meeting records and agendas. The system is based on classification of documents according to contents which enables efficient information retrieval.

Avtomatizacija pisarniškega poslovanja (APP) (office automation) je v zadnjem času deležna velike pozornosti računalniških strokovnjakov in uporabnikov. Razlogov je več. Z avtomatizacijo teh aktivnosti lahko močno povečamo učinkovitost posameznih administrativnih, strokovnih in vodstvenih delavcev na eni strani, na drugi strani pa tudi učinkovitost organizacije kot celote.

Priča smo intenzivnemu razvoju različnih programskih paketov za avtomatizacijo posameznih aktivnosti na tem področju. Analiza razpoložljivih aplikacij oziroma programskih produktov proizvajalcev računalnikov ali drugih proizvajalcev programske opreme pa kaže, da je večina razpoložljivih produktov usmerjena na razmeroma ozek segment obravnavanega področja, to je na področje obdelave tekstov. Na trgu je prava množica različnih urejevalnikov besedil tujih in domačih avtorjev, medtem ko so drugi segmenti avtomatizacije pisarniškega poslovanja, ki so z vidika povečanja učinkovitosti organizacije nedvomno pomembnejši, še razmeroma neobdelani.

Zato smo se odločili, da v sodelovanju z Iskra-Delto in LB-Gospodarsko banko razvijemo programski paket APP-1, ki posega na novo področje APP.

V vseh OZD, še posebno pa v velikih, se skozi organizacijo pretaka velika množica različnih poslovnih dokumentov (ki niso finančno-računovodske narave, le-ti so bolj ali manj sistematično obdelani v okviru finančno-računovodskega podsistema OZD), od katerih je v veliki meri odvisna učinkovitost OZD. V mislih imamo dokumente, ki sodijo bodisi v kategorijo poslovne korespondence, bodisi samoupravne splošne akte, sejna gradiva, zapisnike sej, različne analize, poročila, organizacijske predpise, študije, projektno dokumentacijo itd.

V večjih OZD letno nastane tudi na desettisoče tovrstnih dokumentov, katerih evidentiranje, shranjevanje in iskanje povzroča velike težave.

Zaradi nepreglednosti in nekonsistentnosti klasičnih načinov vzdrževanja poslovnega sistema, prihaja do velikih časovnih izgub pri iskanju dokumentov, zamud pri reševanju posameznih zadev ter neurejenega in neučinkovitega poslovanja OZD kot celote.

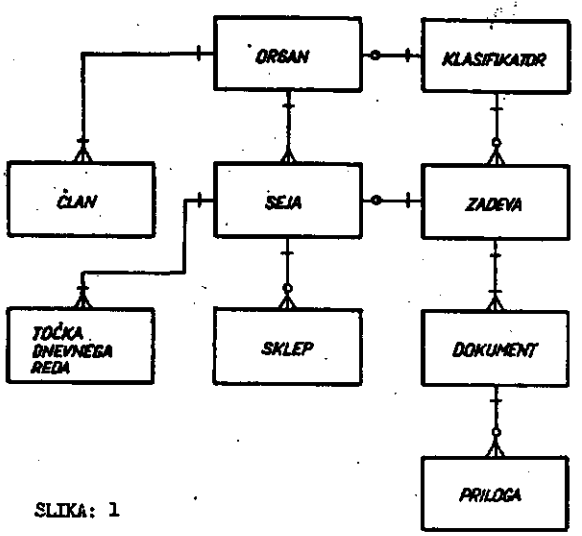
Pri koncipiranju programskega paketa APP-1 smo se naslonili na lastne raziskave obravnavanega področja, dela pa tudi na opise sorodnih programskih paketov iz literature (1), (2).

Zasnova sistema je že na samem začetku zahtevala razrešitev dveh pomembnih vprašanj:
- katere podatke zajeti v sistemu,
- kako organizirati zbirke dokumentarnega gradiva in evidence podatkov o njih, da bo omogočen hiter dostop oziroma iskanje dokumentov.

Odločili smo se, da se računalniško vodijo samo ključni podatki o posameznem dokumentu ne pa kompletni teksti, dokumenti se arhivirajo na klasičen način ali mikrofilmajo.

Raziskava je pokazala, da je v sistemu potrebno zajeti sledeče entitete:
- zadeva (skupina vsebinsko povezanih dokumentov)
- dokument (enota dokumentarnega gradiva)
- priloga (priloga dokumentu)
- organ (delovno telo, komisija, odbor, projektni team, itd.)
- seja (seja oziroma sestanek določenega organa, ki se evidentira in za katerega se pripravi določeno gradivo)
- sklep (sklep določenega organa, ki zahteva neko akcijo, ima rok in odgovorno osebo za izvršitev)
- točko dnevnega reda
- član (člani delovnih teles, komisij, odborov, organov, itd.)
- klasifikator (vsebinsko geslo zadeve).

Entitetni model sistema prikazuje slika 1.



SLIKA: 1

Na ta način je mogoče v sistemu zajeti večino poslovne dokumentacije, ki nastaja v neki organizaciji. Pri izboru atributov navedenih entitet, smo se morali odločiti za kompromisno pot. Ker gre v veliki meri za tekstualne podatke, njihov vnos zahteva veliko časa pa tudi veliko prostora na spominskih medijih. Če želimo, da bo sistem v praksi učinkovit, potem računalniška obdelava posameznega dokumenta oziroma zadeve ne sme zahtevati veliko delovnega časa.

Ob upoštevanju teh robnih pogojev smo prišli do seznama atributov, ki je okvirno za entiteto zadeva sledeč:
- klasifikacijski znak
- zaporedna številka
- letnica
- organizacijska enota
- subjekt
- besedna identifikacija zadeve
- signirni znak
- datum nastanka zadeve
- zveza z drugimi zadevami
- številka mikrofilma
- tajnost
- datum spremembe

Organizacijsko sistem temelji na uporabi klasifikacijskega načrta (urejenega seznama gesel), ki omogoča vsebinsko razvrščanje dokumentov oziroma zadev. Vlogo klasifikacijskega načrta v tem sistemu lahko primerjamo z vlogo kontnega plana v finančnem poslovanju. Vsakemu prispelemu dokumentu oziroma zadevi se pred vnosom v računalnik priredi klasifikacijski znak - vsebinsko geslo, ki pove kam obravnavani dokument, zadeva sodi po vsebini. V skladu s klasifikacijskim načrtom so zadeve tudi fizično urejene v arhivu. Klasifikacijski znak - geslo pa nam služi tudi kot ključ pri poizvedovanjih o zadevah oziroma dokumentih, ki sodijo k neki zadevi.

Programski paket APP-1 je zasnovan tako, da omogoča:
- formiranje klasifikacijskega načrta, ki vsebuje vsebinska gesla za celotno področje poslovanja delovne organizacije
- klasificiranje dokumentov in njihovo razvrščanje po vsebini
- združevanje vsebinsko povezanih dokumentov v zadeve
- formiranje računalniške evidence o zadevah, dokumentih in prilogah dokumentov
- formiranje računalniške evidence o organih in sestavi organov
- evidentiranje sej, točk dnevnega reda in sklepov
- formiranje registra sklepov in kontrola njihovega izvrševanja
- spremljanje nosilcev in rokov za rešitev posameznih zadev

Pomemben del sistema so iskalni mehanizmi, ki omogočajo poizvedovanja (information retrieval) po različnih kriterijih, kot so denimo:
- klasifikacijski znak - vsebinsko geslo zadeve
- šifra zadeve
- subjekt

Sistem bo prirejen za uporabo na računalnikih ISKRE-DELTE in na računalniku DEC-10.

Literatura:

1) Administrative Control System, IBM Corporation - General Systems Division, Datapro Report on Automated Office Software, A50-100-132, 1981

2) An Office Automation Study, Datapro Report on Automated Office Systems, A33-120-101, 1981

3) Elektronic Text Retrieval Systems, Datapro Report on Automated Office Systems, A20-402-101, 1981

4) Storage and Retrieval - How to Store the Paper So You Can Get It Back, Datapro Report on Automated Office Systems, A41-755-101, 1981

5) S.G. Price, Introducing the Electronic Office, NCC Publications, Manchester, 1979

6) Records Management: What to do With the Paper After You Have it. Datapro Report on Automated Office Systems, A41-750-101, 1981

7) H.L. Morgan, Control and Tracking of Office Documents, Decision Sciences Working Paper, University of Pennsylvania, 1978

UPRAVLJANJE PROCESOV

PROCESS CONTROL

MIKRORAČUNALNIŠKA REALIZACIJA REGENERATIVNEGA SLEDENJA V REALNEM ČASU

Andrej DOBNIKAR
Veselko GUŠTIN
Tone VIDMAR
Fakulteta za elektrotehniko
Ljubljana, Tržaška 25
Jugoslavija

UDK: 681.3.012

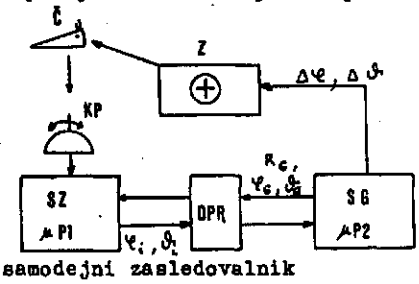
POVZETEK - Prikazan je sistem za regenerativno sledenje v realnem času. Opisana je kinematika problema, koncept programske opreme in aparaturna zgradba. Rezultati simulacije, ki so definirali sistemske parametre, so pokazali kvaliteto sledenja, kakršno smo dosegli tudi pri realnem sistemu.

MICROCOMPUTER REALIZATION FOR REGENERATIVE CONTROL IN REAL TIME
ABSTRACT - The system for regenerative control in real time is shown. Kinematics of the problem is outlined together with the software and hardware organisation. The results of simulation, that enabled us to define system parameters, show quality level of regenerative control. The same level was obtained in real system.

1. Uvod

Za regenerativno sledenje objekta, ki se giblje po trajektoriji v prostoru, je značilno naslednje: 1.) smer gibanja objekta je vnaprej nepoznana, zato jo ugotavljamo v realnem času s pomočjo ročnega zasledovanja; 2.) ročno zasledovanje omogoča izvedbo meritev razdalje ter s tem generiranje vzorcev trajektorije, ki služijo za izračun bodočih točk trajektorije; 3.) izračunavanje bodočih točk trajektorije na osnovi točk, ugotovljenih z ročnim zasledovanjem, predstavlja avtomatično oz. regenerativno zasledovanje gibanja objekta po prostorski krivulji; 4.) samodejno sledenje je mogoče spreminjati z ročnim vnašanjem korekcij.

Slika 1 podaja shematični prikaz problema.



SZ - samodejni zasledovalnik
SG - simulator gibanja objekta
Z - zaslon
KP - krmilna palica
DPR - vmesnik (Dual port RAM)

Sl. 1. Predstavitev problema

Zanjo je značilna zaključena zanka, katere pomemben člen je tudi človek-operater. SZ in SG sta realizirana z M 68000 in Z 80 A mikroračunalnikoma, zaslon Z je izveden s koordinatnim osciloskopom HP, KP pa predstavlja analogni vhod v SZ. Človek (Č), ki zaključuje zanko, na osnovi opazovanja razlike med generiranimi in izračunanimi vrednostmi kotov ($\Delta\varphi$, $\Delta\vartheta$), ki jo prikazuje Z, vpliva preko KP na SZ z namenom, da se ta razlika zmanjša.

Regenerativno sledenje omogoča izključitev človeka iz postopka zasledovanja gibanja objekta po prostorski trajektoriji. Človek je potreben le v začetni fazi, ko z ročnim sledenjem vnaša podatke v SZ. Na osnovi teh podatkov je v nadaljevanju mogoče zasledovanje objekta avtomatično, kar pomeni, da človek tedaj le še korigira eventuelno odstopanje izračunane trajektorije od dejanske. Glede na to lahko celoten postopek regenerativnega sledenja razdelimo v dve fazi: v fazo učenja, v kateri človek z ročnim sledenjem vnaša podatke v sistem, ter v fazo samodejnega sledenja, ko sistem s predikcijo na osnovi znanih točk trajektorije računa nove točke.

V nadaljevanju želimo podrobneje opisati delovanje SZ in SG v obeh omenjenih fazah regenerativnega sledenja. Podana bo tudi njuna aparaturna realizacija z vidika vhodno-izhodnih funkcij, ter monitorska nadgradnja, ki ju medsebojno povezuje v sistem. Končno bo podana še dvojna vloga mikroračunalnika Z 80 A (Partner -

Iskra Delta), ki poleg simulatorja gibanja predstavlja še razvojno postajo oz. aktivni terminal za zagon sistema, za njegovo testiranje ter vnašanje podatkov.

2. Kinematika gibajočega objekta v prostoru

Gibanje objekta v prostoru opazujemo lahko v inercialnem (IN) koordinatnem sistemu ali v neenakomerno vrtečem se opazovalnem sistemu (LOS), ki se vrti skupaj z gibanjem objekta po trajektoriji v prostoru. Na začetku opazovanja sistema IN sovpada s sistemom LOS.

V IN sistemu opazujemo gibanje objekta z radij vektorjem $\vec{R}_{IN}(t)$, hitrostjo $\vec{v}_{IN}(t)$ in pospeškom $\vec{a}_{IN}(t)$, v LOS sistemu pa z $\vec{R}_{LOS}(t)$, $\vec{v}_{LOS}(t)$ in $\vec{a}_{LOS}(t)$. Povezavo med $\vec{R}_{IN}$ in $\vec{R}_{LOS}$, $\vec{v}_{IN}$ in $\vec{v}_{LOS}$ ter $\vec{a}_{IN}$ in $\vec{a}_{LOS}$ določajo enačbe:

$$\left(\frac{d}{dt}\right)_{IN} = \left(\frac{d}{dt}\right)_{LOS} + \vec{\omega} \times$$

$$\left(\frac{d^2}{dt^2}\right)_{IN} = \left[\left(\frac{d}{dt}\right)_{LOS} + \omega\overrightarrow{\times} \left(\frac{d}{dt}\right)_{LOS} + \vec{\omega} \times\right], \quad (1)$$

ki omogočajo izračun relacij med istoimenskimi veličinami obeh koordinatnih sistemov:

$$\vec{R}_{IN} = \vec{R}_{LOS}$$

$$\vec{v}_{IN} = \vec{v}_{LOS} + \vec{\omega} \times \vec{R} \quad (2)$$

$$\vec{a}_{IN} = \vec{a}_{LOS} + 2\vec{\omega} \times \vec{v}_{LOS} + \vec{\omega} \times (\vec{\omega} \times \vec{R}) + \dot{\vec{\omega}} \times \vec{R} ,$$

kjer je: $2\vec{\omega} \times \vec{v}_{LOS}$ - Coriolisov pospešek

$\vec{\omega} \times (\vec{\omega} \times \vec{R})$ - centrifugalni pospešek

$\dot{\vec{\omega}} \times \vec{R}$ - pospešek zaradi neenakomerno se vrtečega LOS sistema

Predpostavimo, da velja:

$$\vec{a}_{IN} = a_\varphi \cdot \vec{i} + a_\vartheta \cdot \vec{j} + a_R \cdot \vec{k} = (a_\varphi , a_\vartheta , a_R)$$

$$\vec{v}_{IN} = v_\varphi \cdot \vec{i} + v_\vartheta \cdot \vec{j} + v_R \cdot \vec{k} = (v_\varphi , v_\vartheta , v_R)$$

$$\vec{R}_{IN} = R \cdot \vec{k} = (0,0,R) \quad (3)$$

in:

$$\vec{\omega}_{LOS} = \omega_\varphi \cdot \vec{i} + \omega_\vartheta \cdot \vec{j} + \omega_R \cdot \vec{k} = (\omega_\varphi , \omega_\vartheta , \omega_R)$$

$$\vec{a}_{LOS} = (0,0,\ddot{R})$$

$$\vec{v}_{LOS} = (0,0,\dot{R})$$

$$\vec{R}_{LOS} = (0,0,R) \quad (4)$$

kjer so i, j, k enotni vektoriji v smeri x, y, z. Če vstavimo (4) v (1) in (2), dobimo:

$$\vec{v}_{IN} = \frac{d\vec{R}}{dt}_{LOS} + \vec{\omega} \times \vec{R} = (\omega_\vartheta \cdot R, -\omega_\varphi \cdot R, \dot{R}) \quad (5)$$

in

$$\vec{a}_{IN} = (2\dot{R} \cdot \omega_\vartheta + R \cdot \omega_\varphi \cdot \omega_R + \dot{\omega}_\vartheta \cdot R ,$$

$$-2\dot{R} \cdot \omega_\varphi + R \cdot \omega_\vartheta \cdot \omega_R - \dot{\omega}_\varphi \cdot R , \quad (6)$$

$$\ddot{R} - (\omega_\varphi^2 + \omega_\vartheta^2) \cdot R)$$

oziroma:

$$v_\varphi = \omega_\vartheta \cdot R$$

$$v_\vartheta = -\omega_\varphi \cdot R$$

$$v_R = \dot{R}$$

$$a_\varphi = 2\dot{R} \cdot \omega_\vartheta + R \cdot \omega_\varphi \cdot \omega_R + \dot{\omega}_\vartheta \cdot R \quad (7)$$

$$a_\vartheta = -2\dot{R} \cdot \omega_\varphi + R \cdot \omega_\vartheta \cdot \omega_R - \dot{\omega}_\varphi \cdot R$$

$$a_R = \ddot{R} - (\omega_\varphi^2 + \omega_\vartheta^2) \cdot R$$

Največkrat pa podajamo gibanje objekta v t.im. globalnem koordinatnem sistemu (G), v katerem je os Z navpična, ostali dve (Y, X) pa vodoravni. Os X leži na projekciji R na vodoravno ravnino, Y pa je pravokoten na ravnino (X x Z). Gibanje objekta v sistemu G podajajo enačbe:

$$\vec{a}_{IN} = a_X \cdot \vec{I} + a_Y \cdot \vec{J} + a_Z \cdot \vec{K}$$

$$a_X = a_R \cdot \cos\vartheta + a_\varphi \cdot \sin\vartheta$$

$$a_Y = -a_\vartheta$$

$$a_Z = a_R \cdot \sin\vartheta + a_\varphi \cdot \cos\vartheta \quad (8)$$

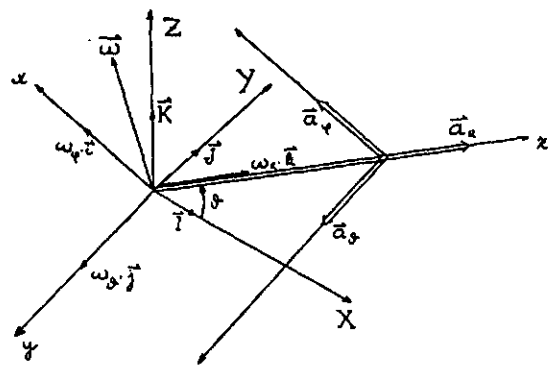
$$\vec{v}_{IN} = v_X \cdot \vec{I} + v_Y \cdot \vec{J} + v_Z \cdot \vec{K}$$

$$v_X = v_R \cdot \cos\vartheta + v_\varphi \cdot \sin\vartheta$$

$$v_Y = -v_\vartheta$$

$$v_Z = v_R \cdot \sin\vartheta + v_\varphi \cdot \cos\vartheta$$

Slika 2 ilustrira zgornje izraze.



Sl. 2. Ilustracija sistemov G(X,Y,Z), IN in LOS(x,y,z)

Ker v sistemu G nastopata le ω_β in ω_ε, moramo tudi komponente $\vec{\omega}$ izraziti drugače:

$$\omega_\varepsilon = \omega_\varepsilon^\beta \cdot \cos\beta$$

$$\omega_R = \omega_\varepsilon^\beta \cdot \sin\beta \qquad (9)$$

$$\omega_\beta = \omega_\beta$$

Tukaj predstavlja ω_ε^β kotno hitrost pri vrtenju okoli navpične osi. Če upoštevamo izraze (9) v (7), dobimo:

$$v_\varepsilon = \omega_\beta \cdot R$$

$$v_\beta = -\omega_\varepsilon^\beta \cdot R\cos$$

$$v_R = \dot{R}$$

$$a_\varepsilon = 2\dot{R}\cdot\omega_\beta + \dot{\omega}_\beta \cdot R + R\cdot\omega_\varepsilon^{\beta^2}\cdot\cos\beta\cdot\sin\beta$$

$$-\frac{a_\beta}{\cos\beta} = 2\dot{R}\cdot\omega_\varepsilon^\beta + \dot{\omega}_\varepsilon^\beta \cdot R - 2\,\omega_\varepsilon^\beta\cdot\omega_\beta\cdot\mathrm{tg}\beta\cdot R$$

$$a_R = \ddot{R} - (\omega_\beta^2 + \omega_\varepsilon^{\beta^2}\cdot\cos^2\beta)\cdot R \qquad (10)$$

Pri izdelavi postopkov za regenerativno sledenje smo predpostavljali $\vec{a}_{IN} = 0$. Tedaj se izrazi (10) poenostavijo v:

$$2\frac{\dot{R}}{R}\cdot\omega_\beta + \dot{\omega}_\beta + \omega_\varepsilon^{\beta^2}\cdot\cos\beta\cdot\cos\varepsilon = 0$$

$$2\frac{\dot{R}}{R}\cdot\omega_\varepsilon^\beta + \dot{\omega}_\varepsilon^\beta - 2\cdot\omega_\varepsilon^\beta\cdot\omega_\beta\cdot\mathrm{tg}\beta = 0 \qquad (11)$$

$$\ddot{R} - (\omega_\varepsilon^{\beta^2}\cdot\cos^2\beta + \omega_\beta^2)\cdot R = 0$$

oziroma:

$$\dot{\omega}_\beta = -2\frac{\dot{R}}{R}\cdot\omega_\beta - \omega_\varepsilon^{\beta^2}\cdot\cos\beta\cdot\sin\beta$$

$$\dot{\omega}_\varepsilon^\beta = 2\cdot\omega_\varepsilon^\beta\cdot\omega_\beta\cdot\mathrm{tg}\beta - 2\frac{\dot{R}}{R}\cdot\omega_\varepsilon^\beta \qquad (12)$$

$$\ddot{R} = (\omega_\varepsilon^{\beta^2}\cdot\cos^2\beta - \dot{\omega}_\beta^2)\cdot R = \Omega^2\cdot R$$

3. SZ - samodejni zasledovalnik (programski del)

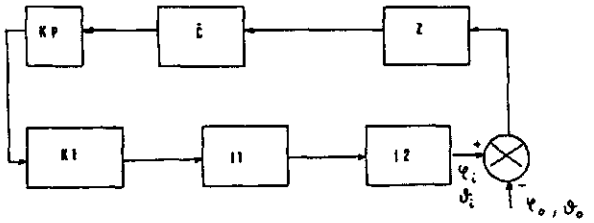
Kot smo omenili že uvodoma, izvaja SZ funkcijo sledenja gibanja objekta v prostoru v dveh fazah: v fazi učenja in fazi samodejnega sledenja.

3.1. Faza učenja

Fazo učenja nadalje razdelimo v način ročnega sledenja (RAM) in način ročnega sledenja z merjenjem razdalje (RAML).

A.) Način ročnega sledenja (RAM):

Blok diagram, ki podaja prvi korak faze učenja, podaja slika 3. V sliki Č določa človeka, ki na osnovi razlike med izračunanimi koti ε_i in β_i ter koti objekta ε_o, β_o, ki jo prikazuje zaslon Z, premika komandno palico KP. Pri simulaciji zgornje zanke smo predpostavili prenosno funkcijo človeka:



Sl. 3. Blok shema ročnega sledenja RAM

$$\check{C} = \frac{e^{-sT_1}}{1+sT_2} \qquad (13)$$

ter idealizirali Z in KP:

$$Z = KP = 1 \qquad (14)$$

K1 je kompenzatorski blok, katerega funkcija je, da kompenzira zakasnitev in inercijo človeka. Njegovo prenosno funkcijo določa enačba:

$$K1 = \frac{A + sB + s^2C}{1 + sD} \qquad (15)$$

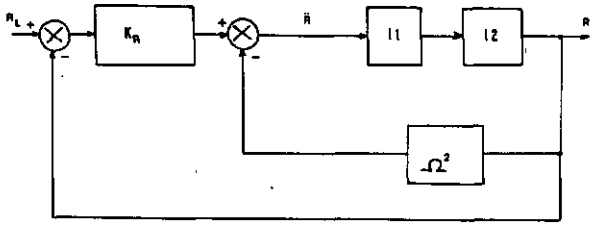
I1 in I2 sta integratorja s prenosnima funkcijama:

$$I1 = \frac{1}{\tau_1 s}$$

$$I2 = \frac{1}{\tau_2 s} \qquad (16)$$

B.) Način ročnega sledenja z merjenjem razdalje (RAML):

RAML korak učenja se od prvega razlikuje po tem, da je v njem aktiven merilec razdalje. Slednji daje s periodo T_L vrednosti R_L, ki so podvržene različnim motnjam. Zaradi tega jih vodimo v R_L filter, katerega blok shemo podaja slika 4.



Sl. 4. R filter za odpravo motenj

K_R je vhodni kompenzator merilnika razdalje s prenosno funkcijo:

$$K_R = 1 + T_L . S ,$$

I1 in I2 sta integracijska bloka (glej (16)), Ω^2 pa izhaja iz enačbe (12).

3.2. Faza samodejnega sledenja

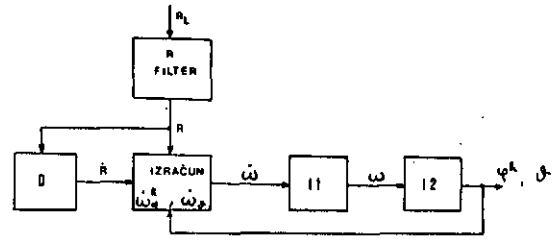
Tudi tukaj ločimo dva različna načina, ki si sledita eden za drugim.

C.) Način regenerativnega sledenja (RC):

Za RC način je značilno računanje ω_φ^h in ω_ϑ na osnovi enačb (12) ter določitev φ^h in ϑ samo s postopkom zaporednih integracij, torej brez vpliva človeka. Blok shemo RC načina prikazuje slika 5. Izračun ω_φ^h, ω_ϑ upošteva predpostavljene vrednosti za $\varphi^h, \vartheta, \omega_\varphi^h, \omega_\vartheta$ na osnovi parabolične regresije, ki posega 10 intervalov nazaj /2/.

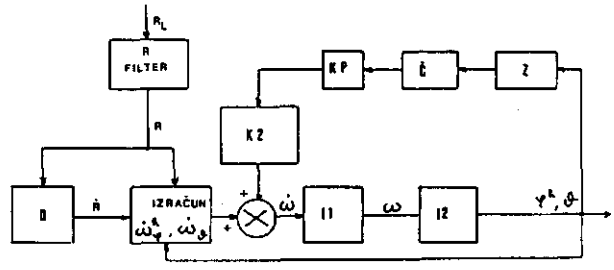
D.) Način regenerativnega sledenja s korigiranjem (RCJ):

Zadnji korak v postopku samodejnega sledenja predstavlja način RCJ, ki se razlikuje od prednega po tem, da upošteva poleg izračunanih pospeškov ω_φ^h in ω_ϑ še korekcije $\omega_{\varphi k}$ in



Sl. 5. Blok shema regenerativnega sledenja

$\omega_{\vartheta k}$, ki jih človek vnaša preko KP z namenom, da korigira sledenje. Blok shemo RCJ podaja slika 6:



Sl. 6. Regenerativno sledenje s korigiranjem RCJ

K2 na zgornji sliki zopet predstavlja blok za kompenzacijo vpliva človeka. Določa ga enačba:

$$K2 = \frac{E + s.F}{G + s.H} \qquad (17)$$

Prehajanje med posameznimi načini je izvedeno deloma programsko, deloma pod kontrolo človeka -operaterja. Načinu A sledi način B ob zahtevi Z1, ki jo vnese operater takrat, ko je ročno sledenje zadovoljivo. Prehod na način C je avtomatičen in se izvede, ko sistem dobi iz R filtra dovolj (3) dobrih zapovrstnih meritev. Sledi samodejno sledenje (C), ki ga zamenja način D ob zahtevi Z2, ki jo zopet vnese operater, ko ugotovi odstopanje izračunanega gibanja objekta.

Digitalno procesiranje vseh štirih načinov, ki tvorijo skupaj SZ, izhaja direktno iz regulacijskih zank na slikah 3 do 6. Posamezne bloke, ki se v zankah nahajajo, je potrebno le prevesti iz frekvenčnega (s) v diskretni (z) prostor. Slednje opravi bilinearna transformacija:

$$s = \frac{2}{T} \frac{(1 - z^{-1})}{(1 + z^{-1})} \qquad (18)$$

kjer je T perioda vzorčenja.

Primer 1: Za R filter na sliki 4 določimo diskretno odvisnost izhoda R od vhodnih vzorcev R_L.

$$\frac{R}{R_L} = \frac{1 + T_L \cdot s}{s^2 - \Omega^2 + 1 + T_L \cdot s} \qquad (19)$$

V enačbi (19) smo predpostavili $\tau_1 = \tau_2 = \tau$. S pomočjo transformacije (18) dobimo:

$$R_i\left(\frac{4\tau^2}{T^2} + \frac{2T_L}{T} + 1 - \Omega^2\right) = R_{i-1}\left(\frac{8\tau^2}{T^2} - 2 + 2\Omega^2\right) + R_{i-2}\cdot\left(\frac{2T_L}{T} - \frac{4\tau^2}{T^2} - 1 + \Omega^2\right) +$$

$$+ R_{L_i}\cdot\left(\frac{2T_L}{T} + 1\right) + R_{L_{i-1}}\cdot 2 + R_{L_{i-2}}\cdot\left(1 - \frac{2T_L}{T}\right)$$

$$R_i = -R_{i-1} + R_i\cdot\frac{2\tau}{T} - R_{i-1}\cdot\frac{2\tau}{T}; \text{ (blok D)} \qquad (20)$$

Parametre, ki se nahajajo v blokih shem (slike 3, 4, 5, 6), smo določili s pomočjo programskega paketa za načrtovanje vodenja ANA /3,4/. Simulacijski rezultati, ki upoštevajo tako dobljene sistemske parametre, so prikazani na slikah 7, 8, 9, 10, 11. V okviru simulacije smo realizirali tudi generator trajektorije, merilec razdalje in naključni generator motenj, ki smo ga vključili pri odčitavanju razdalje.

V diagramih, ki ponazarjajo rezultate simulacije, pomeni:

- φ^h - azimut (ravninski kot)
- ϑ - elevacija (vertikalni kot)
- E_{φ^h} - napaka azimuta (razlika med generiranim in izračunanim azimutom)
- E_ϑ - napaka elevacije
- R - generirana razdalja
- R_L - merjena razdalja z vsebovano naključno motnjo
- $\dot{\varphi}^h = \frac{d\varphi^h}{dt} = \omega_\varphi^h$ - hitrost spremembe azimuta
- $= \frac{d\vartheta}{dt} = \omega_\vartheta$ - hitrost spremembe elevacije
- $E_{\dot{\varphi}^h}$ - napaka v hitrosti spreminjanja azimuta
- $E_{\dot{\vartheta}}$ - napaka v hitrosti spreminjanja elevacije

Rezultati simulacije kažejo, da sta pogreška E_ϑ in E_{φ^h} v načinu D(RCJ) dovolj dolgo (več kot 10 s) v okviru dopustne tolerance (0,01 rad) in to navkljub znatnim motnjam pri meritvi razdalje R_L.

4. Aparaturna oprema sistema za samodejno sledenje

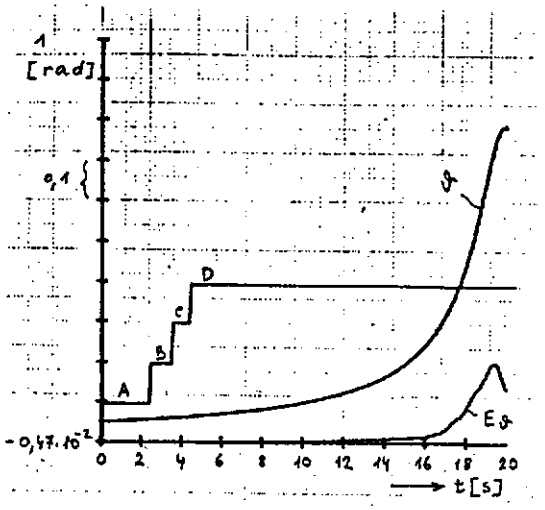
Sistem za regenerativno sledenje v realnem času predstavljata dva računalnika. Prvi računalnik (M 68000) sestavlja nekaj plošč enojnega EVROPA formata in sicer:

a. kartica z mikroprocesorjem M 68000, 4K ROM (s preprostim monitorjem GESBUG), 2K RAM podatkovnega pomnilnika, časovnikom (timerjem) in vmesnikom za RS-232 paralelno serijski prenos /5/;
b. kartica z dodatnim 8K pomnilnikom RAM /5/;
c. paralelni vmesniki PIA, 2 kartici z dvema PIA vezjema /5/;
d. digitalno/analogni vmesnik /5/;
e. analogno/digitalni vmesnik /5/ in
f. kartica DPR.

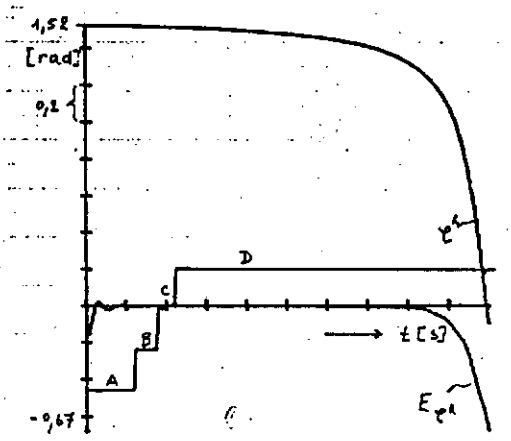
Paralelne vmesnike uporabljamo za krmiljenje lučk in tipk v sistemu. Analogno digitalni pretvorniki nam služijo za odbiranje položajev potenciometrov krmilne palice.

Drugi računalnik je PC Iskra-Partner. Paralelni vmesnik PIO služi za komunikacijo med računalnikoma kot tudi za krmiljenje x in y osi osciloskopa. Blok shemo sistema vidimo na sliki 12.

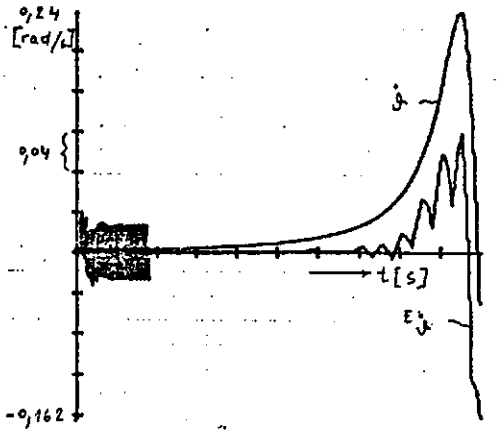
Osnovna zamisel je, da se programska oprema računalnika M 68000, ki v realnem času podpira regenerativno sledenje, ne bi v ničemer razlikovala od realnega sistema. Zato smo s pomočjo DPR plošče realizirali povsem asinhrono povezavo obeh računalnikov. Na plošči DPR (slika 13) je niz pomnilnikov z dvojnim ločenim naključnim dostopom (dual port RAM), ena stran je namenjena M 68000, druga stran pa Partnerju preko PIO. Sinhronizacija poteka le tako, da zadnji zapis niza podatkov v DPR neposredno sproži prekinitev v Partnerju. Le-ta prebrane vrednosti primerja s svojimi izračunanimi (simulirano krivuljo leta) ter razliko preko vmesnika PIO, DPR in DAC 2 prikaže na osciloskopu OSC. Povemo naj, da smo si pomagali z osciloskopom zato, ker nam obstoječa resolucija PC Partnerja ni zadoščala, pa tudi krmilni čas zaslona Partnerja presega čas osnovne zanke računanja, t.j. 20 ms.



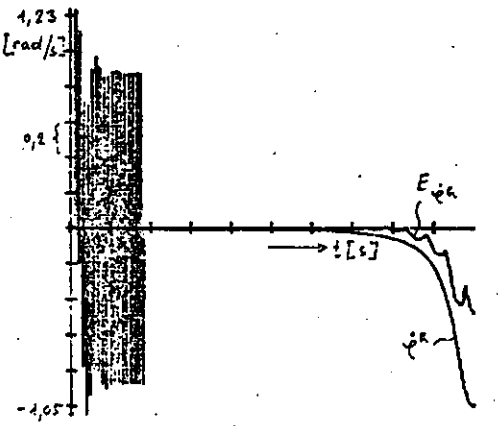
Slika 7.



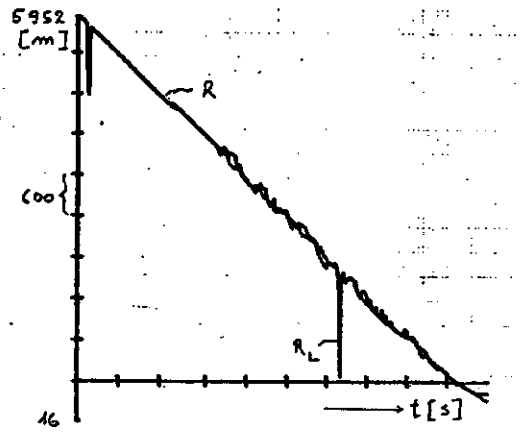
Slika 8.



Slika 9.

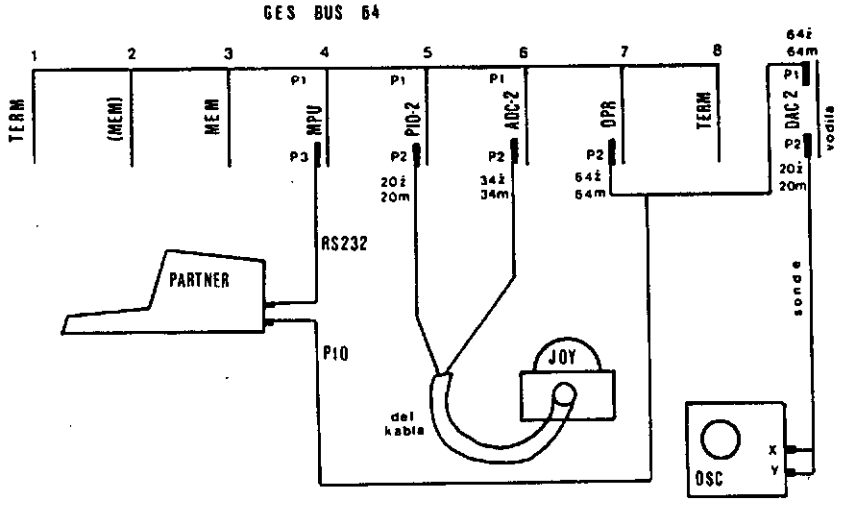


Slika 10.

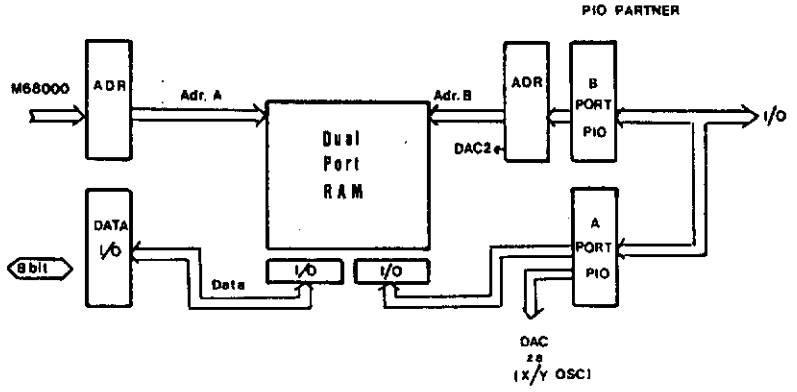


Slika 11.

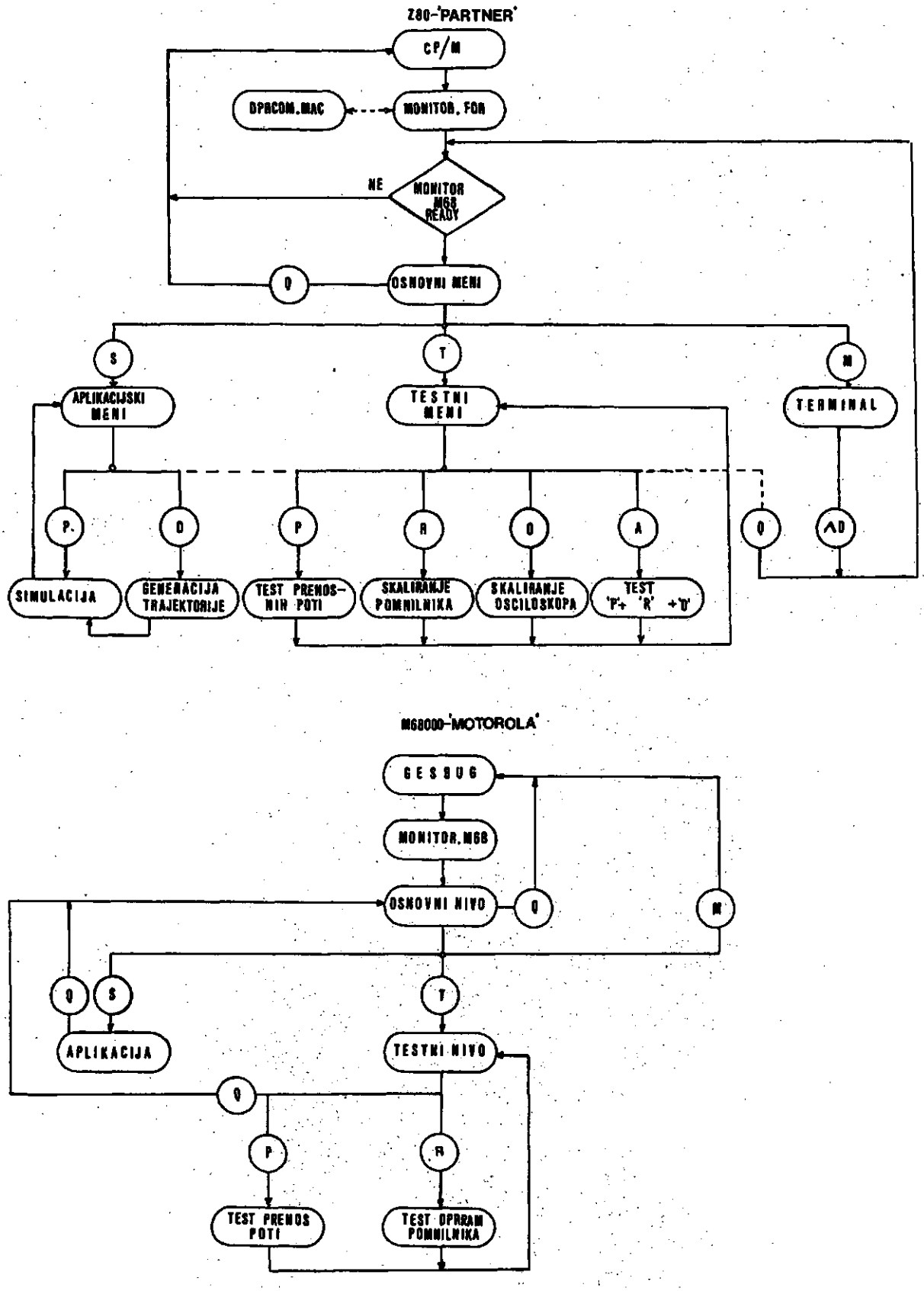
TRENAŽER



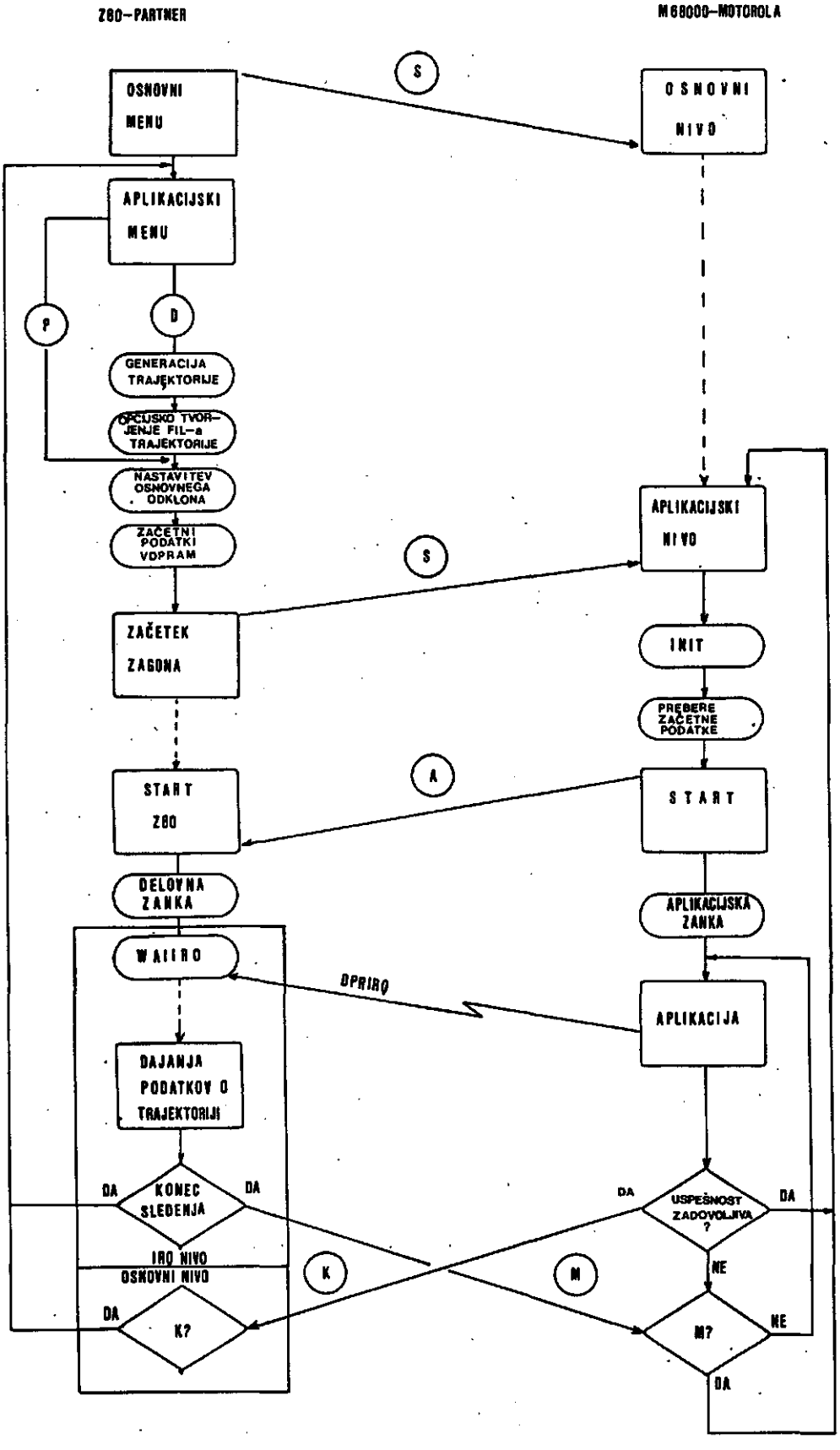
Slika 12. Sistem za regenerativno sledenje



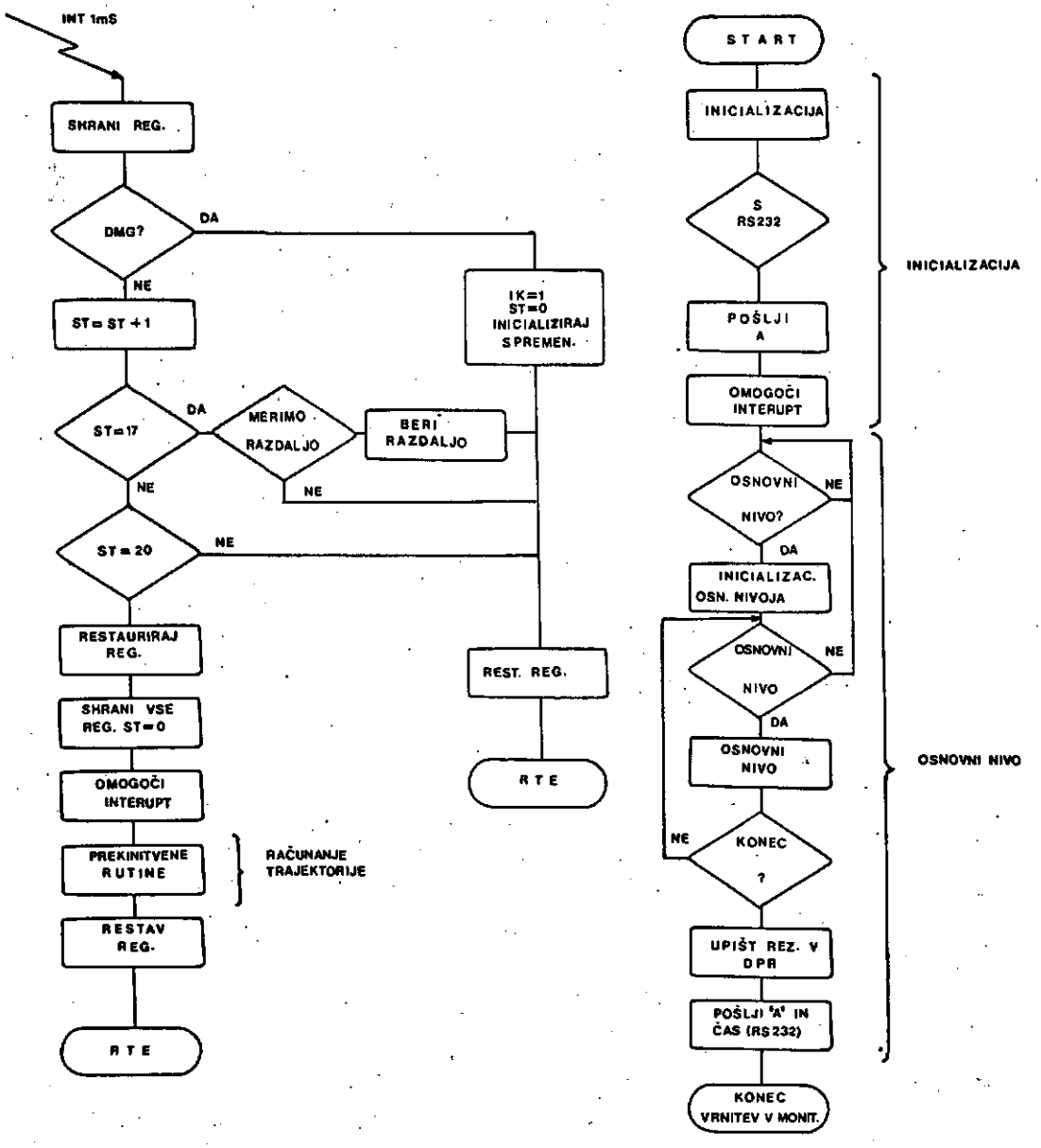
Slika 13. Blok shema plošče DPR



Slika 14. Programska paketa za Z 80 in M 68000



Slika 15. Komunikacijski dialog zagona aplikacije



Slika 16. Aplikativna programska oprema

5. Programska podpora aplikativnemu delu sistema

Omejili se bomo na opis programske opreme, ki podpira aparaturno opremo na nivoju perifernih krmilnikov ter zagotavlja uporabniku "prijazen" dostop do sistema (sistemi "MONITOR").

5.1. Periferni krmilniki

Periferni krmilniki so v sistem vgrajeni kot primitivi, ki se kličejo iz uporabniških programov. Klic je možen iz asemblerskega in fortranskega nivoja. Njihova osnovna naloga je, da omogočajo osnovno komunikacijo med podsistemi (Partner, M 68000, Osciloskop).

"Partner" je povezan na ostale enote sistema preko:

- kanala RS-232 na sistem M 68000,
- dvovhodnega RAM pomnilnika (DPRRAM) na sistem M 68000,
- perifernega busnega sistema na DAC-enoto, ki krmili osciloskop.

Če je potrebna katerakoli od naštetih povezav, je uporabniškemu paketu potrebno priključiti datoteko DPRCOM.MAC.

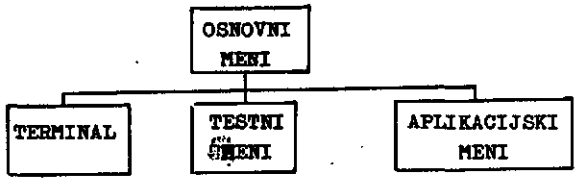
5.2. Nadzorni del (MONITOR)

Programski paket MONITOR je sestavljen iz dveh bistvenih delov:

1 - MONITOR.POR je paket, ki teče na Partnerju
2 - MONITOR.M68 je paket, ki teče na M 68000.

Paketa sta med seboj "simetrična" in sta v stalni medsebojni interakciji (glej sliko 14).

Uporabnik komunicira s sistemom preko menijev. Vhodno enoto predstavlja "Partner". Predpostavlja se, da je zagon sistema uspešno končan. Na zaslonu se prikaže osnovni meni, ki loči med seboj osnovne načine delovanja sistema.



TERMINAL : Partner simulira delovanje asinhronega terminala, Motorola M 68000 se vrne pod nadzor GESBUG firmwarskega paketa. Preden zapustimo ta način, moramo ponovno startati program MONITOR.M68

TESTNI MENI : Omogoča start posameznih testov, ki se izvršijo po zagonu sistema. Po tem lahko z veliko verjetnostjo privzamemo opisano opremo, pripravljeno za operativno delo.

APLIKACIJSKI MENI: Omogoča zagon enega teka sledenja. Možno je tudi generirati nove trajektorije in jih shranjevati na file.

5.3. Zagon

Vsebinsko in časovno sekvenco zagona prikazuje slika 15. Vmesne komunikacije potekajo po kanalu RS-232 in so potrebne ob zagonu in zaključitvi sledenja. Sistem M 68000 kot master sistem sinhronizira medsebojno interakcijo na interruptnem nivoju, ko iz DPRRAM-a prebere parametre naslednje točke trajektorije in, v določenih fazah, tudi podatke merilca razdalje. Sistem Partner, ki simulira okolico, v tem času dobi podatke, ki jih potrebuje za krmiljenje grafičnega izhoda sistema (osciloskopa).

5.4. Organizacija aplikativne programske opreme

Pri zagonu programa se najprej inicializirajo periferne enote in postavi se časovnik. Časovnik se sprogramira kot delilec, tako da sproži prekinitev na 6 nivoju vsako ms.

Pri vsaki prekinitvi se shranijo registri in poveča se števec, ki šteje po modulu 20.

Organizacijo aplikativne programske opreme prikazuje slika 16.

LITERATURA

1. A.B.Chammas, G.J.Friedman: The application of rate-aiding to electro-optical fire control systems.

2. A.Dobnikar, V.Guštin, B.Jurčič-Zlobec, D.Kalanj, A.Likar, D.Matko, B.Orel, M.Šega, T. Vidmar, N.Zimic (po abecednem redu): Interna delovna poročila, P1 - P10, FE S-242, Ljubljana 1985.

3. Matko Drago s sodelavci: Računalniško vodenje dinamičnih procesov in programabilna procesna avtomatika, RSS, URP-C2-0128-781-84.

4. Šega Marko: Računalniško podprta analiza in načrtovanje vodenja dinamičnih sistemov. Magistrska naloga, FE, december 1984.

5. - GESMPU-4A; High Performance 68000 Processor Module (16 bits), Gespac S.A. 1983
- GESMEM-2A; Universal RAM/ROM Memory Module 256 K bytes (16 bits), Gespac S.A. 1983
- GESPIA-2A; Double Parallele Interface Module (2 x 16 I/O), Gespac S.A. 1984
- GESDAC-2; 8 Channels Digital to Analog Converter Module (12 bits resolution), Gespac S.A. 1983
- GESADC-2; 16 Channels, 12 bits Data Acquisition Module, Gespac S.A. 1983
- GESBUG-68; vers. 2.00

6. OPAL-68000 Cross-Assembler 1.03(C) - 1984. Wilke/IDA-Software.

JEZIKOVNO MODELIRANJE PROCESOV

Franc JURKOVIČ, Dali ĐONLAGIĆ, Boris TOVORNIK
UNIVERZA V MARIBORU
TEHNIŠKA FAKULTETA
VTO ELEKTROTEHNIKA, RAČUNALNIŠTVO, INFORMATIKA
62000 M A R I B O R
Smetanova 17
JUGOSLAVIJA

UDK: 681.3.517

Podajamo kratek pregled problematike jezikovnega modeliranja, ki je del področja teorije zamegljenih množic. Opisujemo prehod iz modela, ki ga je opisal Tong, v model, ki sta ga opisala Braae in Rutherford in obratno.

LINGUISTIC MODELLING OF PROCESSES

We give a short review of the linguistic modelling, which is a part ofthe fuzzy sets theory. This is a description of a model transformation, from the model described by Tong into the model described by Braae and Rutherford and inverse.

UVOD

Jezikovni (lingvistični) modeli procesov spadajo v področje teorije zamegljenih množic (fuzzy sets), ki jo je vpeljal L.A. Zadeh 1965. leta /1/. Teorija zamegljenih množic je posplošitev običajne teorije množic. Običajno imamo dve možnosti, reč pripada ali pa ne pripada množici. Pri zamegljenih množicah je pojem pripadnosti razširjen in možno je neskončno število stopenj pripadnosti elementa množici. Pripadnost je določena s pripadnostno funkcijo, ki je lahko normirana in zavzame vrednosti med 0 (nepripadnost) in 1 (popolna pripadnost množici). V neposredni zvezi z zamegljenimi množicami je pojem jezikovne spremenljivke, ki ga je prav tako vpeljal L.A. Zadeh /1/. Ko govorimo, uporabljamo pojme; na primer: "majhno", "srednje", "veliko", ki niso točno definirani, pa jih kljub temu dobro razumemo. Jezikovna spremenljivka je definirana kot spremenljivka, katere vrednosti so stavki v naravnem ali umetnem jeziku.

Modeliranje procesov je hkrati z identifikacijo, kot enim izmed možnih postopkov za realizacijo modelov procesov, temelj moderne teorije upravljanja procesov. Sinteza regulatorjev sloni na poznavanju modelov procesov. Moderna teorija upravljanja procesov je imela velik uspeh na področjih, kjer so modeli procesov dobro poznani. Težave pa nastopajo pri določeni vrsti procesov (na primer kemični procesi), kjer postanejo poznane oblike modelov kompliciranе in praktično neuporabne. Take procese upravlja človek (operater), ki lahko poda jezikovni opis modela procesa (tudi dinamično obnašanje). Do jezikovnega dinamičnega modela procesa pa lahko pridemo tudi s postopkom identifikacije, kot je to prikazal R. Tong /3/. Pri jezikovnih modelih igra važno vlogo njihova struktura. zasledili smo več predlaganih struktur. Možen je prehod (transformacija) iz modela, ki ga je opisal Tong, v model, ki sta ga opisala Braae in Rutherford /2/ in obratno.

ANALIZA STRUKTUR ZAMEGLJENIH MODELOV

Model (A), ki ga je podal Tong /3/ je prikazan na sliki 1, model (B), ki sta ga podala Braae in Rutherford /2/ pa na sliki 2. Oznake so razvidne iz slike 3.Modela sta podana v obliki matrike. Njeni elementi so jezikovne spremenljivke (na primer model A: y(k) je velik, če je y(k-1) majhen in u(k-1) velik).

Prehod iz modela A v model B je sledeč: spremembo izhoda ($\dot{y}$) dobimo z izrazom:

$$\dot{y}(k) = \frac{y(k) - y(k-1)}{T} \qquad (1)$$

k: diskretni trenutek
T: čas vzorčenja

Naslednji korak je zameglitev izraza (1), prevedba na jezikovne spremenljivke.

Model B dobimo iz modela A z zameglitvijo

izraza:

$$y(k) = y(k-1) + T \cdot \dot{y}(k-1) \qquad (2)$$

Omeniti moramo, da smo uporabili za model B proces 1. reda, ker je originalni model B v delu (2) odvisen še od reda procesa, kar pa je po našem mnenju v nasprotju z idejami zamegljenosti.

ZAKLJUČEK

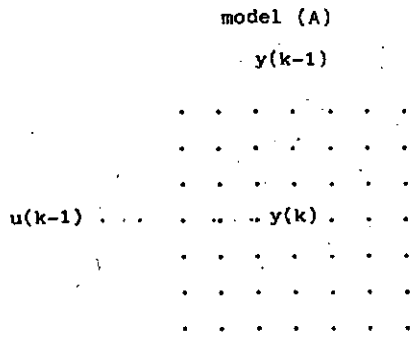
Značilnost modela, ki sta ga opisala Braae in Rutherford, je njegova neodvisnost od časa vzorčenja, medtem, ko je Tongov model neposredno odvisen od časa vzorčenja. Jezikovni modeli procesov so osnova za sintezo jezikovnih regulatorjev, ki nadomestijo človeka pri upravljanju omenjenih procesov. Na razpolago imamo že komercialno dosegljive jezikovne regulatorje (zamegljeni regulatorji). Pokazalo se je, da je raziskava jezikovnih modelov umestna. Samo področje je relativno novo in je v polnem razvoju.

LITERATURA

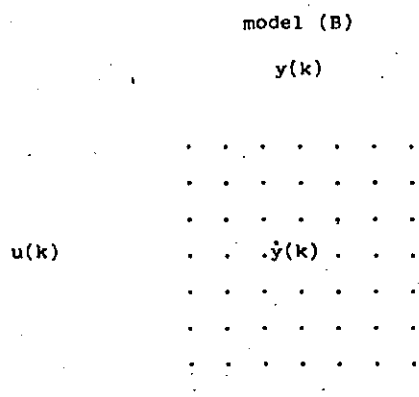
1/ D. D u b o i s, H. P r a d e, Fuzzy sets and systems, theory and applications, Academic press, New York, 1980

2/ M. B r a a e, D. A. R u t h e r f o r d, Theoretical and linguistic aspects of the fuzzy logic controller, Automatica (IFAC), vol. 15, 1979

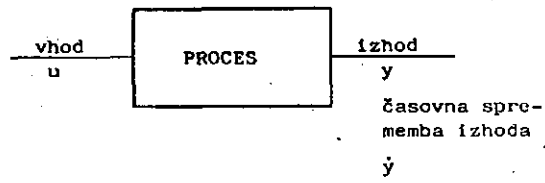
3/ R. M. T o n g, Synthesis of fuzzy models for industrial processes - some recent results, Int. J. General systems, vol. 4, 1978



Slika 1



Slika 2



u, y, ẏ: jezikovne spremenljivke

Slika 3

PRIMENA RACUNARA U IZRADI I KORISCENJU PROGNOSTICKIH MODELA

Dubravka Jakovljevic dipl.matematicar
Institut za vodoprivredu "Jaroslav Cerni", BEOGRAD, JUGOSLAVIJA

UDK: 681.3.02

Prikazani paket programa za realizaciju prognostickog modela "Sava" razradjen je za potrebe izdavanja kratkorocnih prognoza proticaja reke Save u akumulaciju HE Djerdap. Model "Sava" zasnovan je na metodi linearne regresije. Prognoze dobijene radom programa omogucuju rad modela za upravljanje vodnim resursima u realnom vremenu.

USE OF COMPUTERS IN THE DEVELOPMENT AND IMPLEMENTATION OF FORECASTING MODELS

The programme package presented in this paper and used for the implementation of the "Sava" forecasting model was developed for the needs of giving out short-term forecasts of the discharges of the river Sava into the reservior of Iron Gate hydroelectric power plant. The "Sava" model is based on the method of linear regression. The forecasts obtained by using this programme enable the application of the model for managing water resourses in real time.

Ovaj rad razmatra znacaj prognostickih modela u razvoju upravljackih modela, njihovu uzajamnu povezanost, kao i mogucnost primene nove tehnologije u povecanju efikasnosti u upravljanju vodnim resursima.

Poznato je da prognosticki modeli predstavljaju osnovu za rad i razvoj upravljackih modela. Tacna i pravovremena prognoza buduceg stanja sistema omogucava pravilno donosenje odluke o upravljanju sistemom u realnom vremenu. Zato je za svaki ovakav sistem veoma vazno da postoji dobro razvijena mreza stanica na kojima ce biti prikupljeni podaci o vodostajima, proticajima, padavinama, itd. koji omogucavaju pracenje stanja sistema, kao i dobro razvijen prognosticki model za sto bolje definisanje buducih parametara sistema, pa samim tim upravljanje radom sistema u realnom vremenu. Naime, zbog velike kolicine podataka koji se razmatraju kao i potrebe stalnog pracenja stanja sistema i izdavanja prognoza, rad ovakvog sistema tesko bi bio izvodljiv bez primene automatske obrade podataka.

MODEL SAVA

Prognosticki model "Sava" razradjen je u okviru istrazivackog programa za upravljanje rezimom rada HE Djerdap i predstavlja jedan iz kompleksa prognostickih modela za prognozu proticaja na pritokama koje se ulivaju u zoni usora. Cilj izrade ovog modela bio je da se omoguci kontinualno izdavanje kratkorocnih prognoza dotoka Save u akumulaciju HE Djerdap na bazi poznatih hidrometeoroskih informacija u trenutku izdavanja prognoze.

Analizom rezultata dobijenih primenom razlicitih prognostickih modela, zakljuceno je da model LR-linearne regresije daje najprihvatljivije rezultate sa stanovista kvaliteta izdatih prognoza i njihove pouzdanosti. Razmatrani prognosticki model obuhvata identifikaciju i verifikaciju parametara bo,bJk sledece funkcionalne zavisnosti :

$$Q(t) = \sum_{j=1}^{J} \sum_{k=1}^{K} bJk * Q(t-tJk) + b0 \quad (1)$$

gde su :

- Q(t) - prognozirana vrednost u trenutku t
- Q(t-tJk) - osmotrene vrednosti proticaja na J-tom ulaznom profilu u trenutku t-tJk
- bJk,b0 - nepoznati koeficijenti linearne regresije
- J - indeks vodomerne stanice ciji se podaci koriste kao nezavisno promenljive (j=1,J)
- k - indeks vremenskog pomaka tJk, j=1,J, k=1,K

Medjutim, kako je veza izmedju nezavisno i zavisno promenljivih kod hidroloskih prognoza nije linearna, to se prognosticka zavisnist moze pretstaviti u obliku :

$$Q(t) = \prod_{j=1}^{J} \prod_{k=1}^{K} b0 * Q(t-tJk)^{bJk} \quad (2)$$

Da bi odredili parametre bjk (j=1,J,k=1,K) i b0 u jednacini (2), ovu jednacinu logaritmovanjem svodimo na oblik jednacine (1) tj. na oblik pogodan za realizaciju metoda linearne regresije.

$$\log Q(t) = \log b0 + \sum_{j=1}^{J} \sum_{k=1}^{K} bjk * \log Q(t-tjk)$$

Prognosticki model 'Sava' daje jednodnevnu, dvodnevnu, trodnevnu i cetvorodnevnu prognozu proticaja reke Save kod Sremske Mitrovice. Kljucni parametar na osnovu kojeg je izvrsen izbor relevantnih stanica za izdavanje prognoze bilo je vreme doticanja izmedju ulaznih profila i izlaznog profila, za koji izdaje prognoza. U proracunima su korisceni podaci o proticajima na svim izvestajnim stanicama hidrometeroloske sluzbe Jugoslavije na uzvodnom delu sliva reke Save u odnosu na profil vodomerne stanice Sremska Mitrovica." Ove vodomerne stanice su na reci Savi - Mackovac, Slavonski Brod i Zupanja, na Vrbasu - Delibasino Selo, na Bosni - Doboj i na Drini Bajina Basta i Zvornik.

Paket programa 'SAVA' razradjen za prognozu hidrograma na profilu vodomerne stanice Sremska Mitrovica koriscen je i u drugim slicnim istrazivackim programima.

Tako je, na primer, koriscen i u linearnom regresionom modelu za propagaciju talasa kroz vodotok u okviru prognostickog modela 'Gornja Sava'. U ovom modelu, pored niza vrednosti proticaja izmerenih na odredjenim vodomernim stanicama, kao ulazni podaci za model korisceni su i podaci o padavinama izmereni na sektoru Gornja Sava. Medjutim, zbog fizickih karakteristika sliva i prilicno slabe pokrivenosti slivnog podrucja izvestajnim padavinskim stanicama rezultati dobijeni u ovom modelu su znatno losiji, u smislu vremenskog pomeranja hidrograma od onih koje je dao prognosticki model 'Sava'.

Na osnovu ovoga zakljuceno je da je jedna od bitnih predpostavki za dobijanje preciznih prognoza dovoljno gusta mreza osmatrackih stanica, snabdevenih preciznim, savremenim mernim instrumentima, kao i razvijen sistem prenosa informacija od stanica od centra obrade.

Paket programa razradjen u okviru prognostickog modela 'Sava' obuhvata programe za identifikaciju parametara modela, njihovu verifikaciju i korekciju dobijenih rezultata.

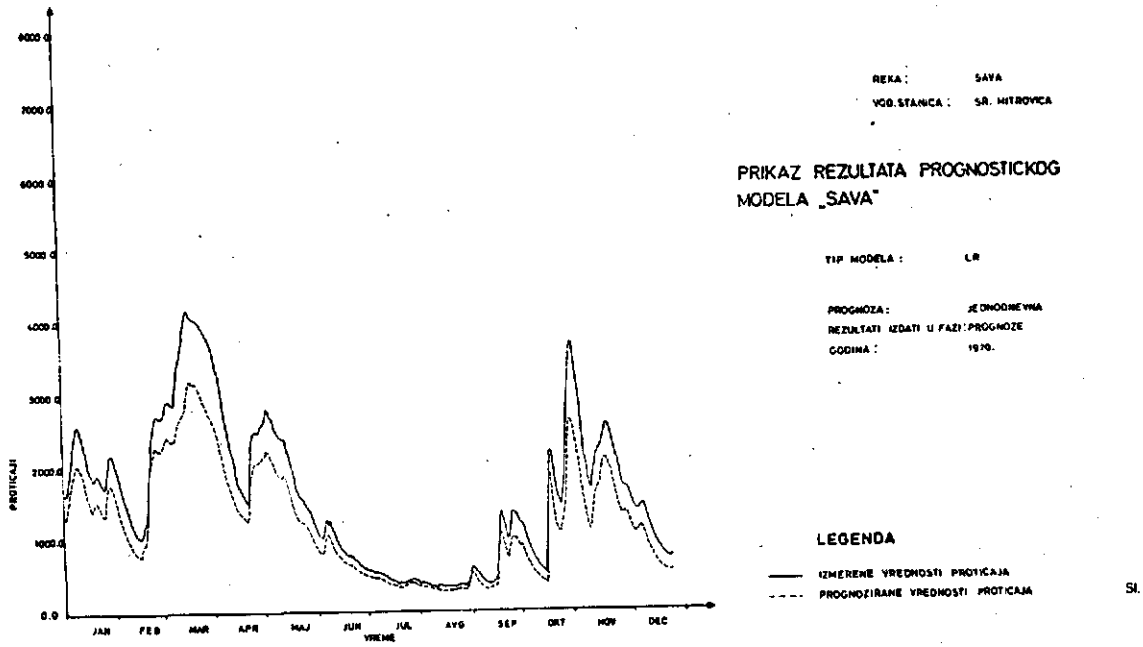
OPIS PODATAKA

Podaci koji se koriste za izvrsenje ovih programa ili se dobijaju kao rezultati proracuna smesteni su u datoteke. Ove datoteke organizovane su sekvencijalno i smestene na jedinstvenom memorijskom medijumu - magnetnom disku.

Da bi se shvatio obim i vrsta informacija neophodnih za realizaciju ovog modela, ovde se u najkracim crtama daje klasifikacija i opis datoteka.

Prema nacinu nastanka datoteke mozemo podeliti u dve grupe - na osnovne i izvedene datoteke.

OSNOVNE DATOTEKE sadrze podatke dobijene merenjem u osmatrackim stanicama i obuhvataju sve relevantne podatke za prvu fazu i deo podataka za ostale faze rada modela. Ove datoteke dalje se mogu podeliti na dve grupe : DATOTEKE FIZICKIH PODATAKA i ISTORIJSKE DATOTEKE DNEVNIH PODATAKA.



Sl. 1

Datoteke fizickih podataka sadrze podatke o fizickim karakteristikama recnih slivova i to :

- podaci o poprecnim i uzduznim profilima reka, odnosno topoloski i morfoloski podaci
- podaci o rapavosti i obraslosti recnog korita

Istorijske datoteke dnevnih vrednosti sadrze sledece podatke :

- vodostaje
- proticaje
- padavine

Prema vremenskom periodu koji se posmatra, istorijske datoteke dnevnih vrednosti dele se u dve grupe :

- datoteke koje sadrze niz dnevnih vrednosti izmerenih za odredjeni period godina, pri cemu su, kad god je to moguce ovim obuhvaceni svi godisnji podaci

- datoteke koje za odredjeni niz godina sadrze podatke samo za pojedine vremenske intervale, pri cemu ovi vremenski intervali ne moraju obavezno pripadati jednoj godini.

Rezultati proracuna smestaju se u izvedene datoteke. Ove datoteke se mogu cuvati u banci podataka ili stampati za upotrebu ili evidenciju. Datoteke koje se cuvaju u banci podataka najcesce su istog oblika kao ulazne datoteke i sadrze podatke potrebne za dalje proracune.

OPIS PROGRAMA

Glavni deo softverske podrske paketu programa 'Sava' predstavlja komandna procedura za interaktivni rad koja omogucava komunikaciju izmedju korisnika, banke podataka i programa. Prikaz rada procedure dat je shemom procedure (slika 2).

Direktnim postavljanjem pitanja preko terminala korisnik bira fazu proracuna (identifikaciju, verifikaviju, proveru ili korekciju) , odnosno program koji zeli da aktivira i ulazne datoteke potrebne za izvrsenje te faze proracuna.

Procedura pocinje sa radom izborom profila, odnosno vodomerne stanice, za koju se vrsi prognoza i izborom vodomernih i kisomernih stanica na kojima su izmerene vrednosti hidrometeoroloskih podataka, potrebne za postupak identifikacije ili verifikacije parametara modela. Pored naziva osmatrackih stanica korisnik procedure mora znati i nazive datoteka u kojima su smesteni odgovarajuci podaci, u banci podataka. Korisniku procedure ostavljeno je da odluci da li ce odabrani program vrsiti linearnu ili nelinearnu regresionu analizu. Osim toga on bira i duzinu vremenskog perioda za koji ce se vrsiti prognoza, odnosno duzinu pomaka izmedju vremenskih serija podataka, na svakom od profila. Sve ove odluke unose se interaktivno u toku izvrsenja procedure.

Prva faza rada obuhvata identifikaciju parametara modela, na osnovu pozantih, izmerenih vrednosti na svim navedenim vodomernim stanicama ukljucujuci i onu za koji se vrsi prognoza. Ulazni podaci za proracun mogu se odnositi na niz godina, i tada su to izmerene dnevne vrednosti proticaja ili padavina, ili na kraci vremenski interval (od desetak dana) u kojima se posmatra neki poplavni talas. Rezultate dobijene proracunom, program za identifikaciju smesta u tri izlazne datoteke.

- Prva datoteka sadrzi tabelu ulaznih podataka i rezultata identifikacje i omogucava lakse pracenje i analizu dobijenih vrednosti.
- Druga datoteka sadrzi niz vrednosti proticaja koji bi se dobili prognozom, u izabranom periodu, ako bi se usvojile vredvosti parametara odredjene identifikacijom. Ovi podaci porede se sa stvarnim vrednostima izmerenim u zadatom periodu. Cilj modela je da odredi takve vrednosti patametara b0, bJk (J=1,J,k=1,K) za koje ce razlika izmedju prognoziranih i izmerenih vrednosti bito sto je moguce manja. Ova datoteka je istog oblika kao osnovne datoteke proticaja i koristi se za graficko predstavljanje rezultata na ploteru (slika 1).
- Treca datoteka sadrzi niz vrednosti parametara modela i predstavlja ulaznu datoteku za program koji vrsi verifikaciju ovih vrednosti.

Identifikacija PARAMETARA na bazi linearne (nelinearne) regresije, izvrsena je na osnovu podataka osmatranja registrovanih u periodu 1961.-1970. godine na navedenim vodomernim stanicama.

Druga faza rada procedure obuhvata verifikaciju parametara dobijenih identifikacijom na osnovu poznatih vrednosti proticaja i padavina na izabranim stanicama, za neki drugi vremenski period.

Verifikacija modela, odnosno utvrdjivanje parametara koji daju najbolju prognozu, radjena je na bazi podataka registrovanih u periodu 1971.-1974. godine i rezultata dobijenih u fazi identifikacije parametara.

Prema tome, prve dve faze imaju zadatak odredjivanje i potvrdjivanje parametara modela, sa ciljem sto veceg poklapanja prognoziranih sa stvarnim vrednostima. One se u toku izvrsenja ove procedure mogu, ukoliko je potrebno, vise puta uzastopno izvrsiti, sve do dobijanja zadovoljavajucih rezultata.

Treca faza rada procedure obuhvata proveru dobijenih vrednosti pametara modela.

Analizom izdatih prognoza u periodima identifikacije, verifikacije i provere i njihovim uporedjivanjem sa registrovanim vrednostima ustanovljeno je da se u izvesnom broju slucajeva pojavljuju greske u smislu vremenskog pomeranja hidrograma u oba smera. Da bi se ove greske eliminisale ili smanjile uvedena je tekuca korekcija prognoze, odnosno parametara modela. U ovom modelu primenjena je korekcija, koji se sastoji u redukciji prognozirane vrednosti sa odgovarjucom greskom prognoze koja je evidentirana u tekucem danu.

U okviru razrade ovog prognostickog modela uradjeni su i programi za crtanje rezultata prognoze, odnosno verifikacije, za sve aspekte ovog modela (slika 1).

Svi programi opisanog paketa programa napisani su na programskom jeziku FORTRAN 77 i realizovani na racunskom sistemu VAX 11/780 operativni sistem VAX/VMS verzija 2.3.

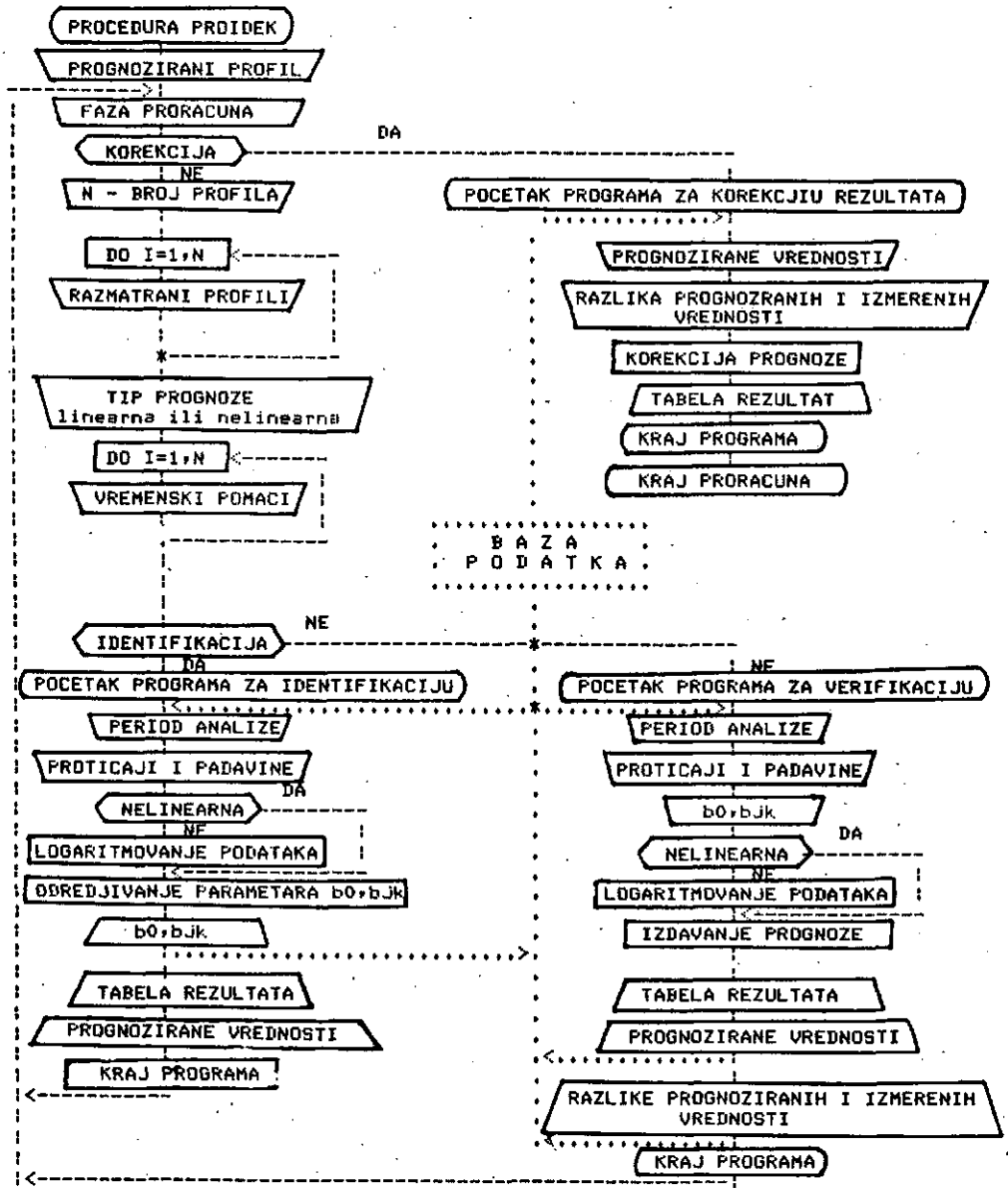
PRAVCI RAZVOJA

Buduci razvoji ovog prognosickog modela kao osnove za razvoj upravljackih modela moze se odvijati u vise pravaca. Izdvoicu samo dva najvaznija

- dalji razvoj prognostickog modela sa ciljem dobijnja prognoza koje boje odrazavaju realno stanje sistema. U nastavku istrazivanja na razvoju prognostickog modela "Sava", veca pazna bice posvecena analizi i adaptaciji parametara modela.

- drugi, mozda i vazniji pravac razvoja, je brza i korektna evidencija informacija na terenu (sto predpostavlja precizne merne instrumente), brz prenos informacija do centra obrade, a zatim i sama obrada. Obzirom da se radi prognostickom modelu koji je osnova upravljackom modelu faktor vreme je ovde najbitniji. Ocekuje se da nova tehnologija u evidentiranju i prenosu informacija do centra obrade omogući sto vecu efikasnost ovog modela.

S H E M A P R O C E D U R E



sl. 2.

PROCESORJI V ISKRINIH TELEFONSKIH SPC CENTRALAH

ŠUBIC MIHAEL, Iskra Telematika Kranj

UDK: 681.3:621.395

Telefonske centrale v svojem razvoju spremljajo elektroniko in izkoriščajo nove dosežke na tem področju. Ob povečanju zahtevnosti krmiljenja, so ga prevzeli procesorji. V 70-tih letih so tovrstne centrale prišle v proizvodnjo tudi v Iskri. Prispevek daje kratek pogled procesorjev, ki jih krmilijo Iskrine centrale.

Telephone exchanges follow the development of the electronics and explont the achievements in this field. The complexity of the exchange control was increased and so it had to be implemented by the processors Iskra started the production of the stored program controlled exchanges in the early seventies. The paper shows an overview of the procesors used in the control of the Iskra's exchanges.

1. VLOGA PROCESORJA V SPC CENTRALI

Telefonske centrale služijo za vzpostavljanje zveze (komutiranje) med poljubnim parom priključnih točk (terminalno vezje). Informacijo o zahtevani zvezi podaja izvorni terminalno vezje v okviru določene signalizacije. Ker je takih terminalnih vezij v centrali tudi po več tisoč in predstavljajo po pravilu med 50% in 85% celotne opreme centrale, je njihova aktivnost reducirana na najnujnejše, to je na prilagoditev nivojev enosmernih in izmeničnih signalov. Terminalna vezja so po eni strani grupirana po signalizacijah (vrstah terminalov ali central na drugi strani prenosnega sistema) po drugi pa po smereh. Vsaka centrala je namreč povezana s skupino terminalnih vezij enake ali različne signalizacije na več central in skupin terminalov.

V SPC (Stored Programm Controled - programsko krmiljenih) central opravljajo terminalna vezja pretvorbo fizičnih signalov v procesorju razpoznavno informacijo. Pri tem ostane ta informacija "onesnažena" z motnjami, napačnimi manipulacijami ter napakami v drugih centralah in terminalih. Od procesorja prejeto informacijo (v obliki izvršilnih ukazov) ista vezja posredujejo proti terminalu ali drugi centrali v obliki, ki ustreza na tem prenosnem sistemu predpisanem protokolu.

Procesor je tako zadolžen za čiščenje informacije iz terminalnih vezij, za njihovo pretvorbo v telefonsko definirane signale in ukrepanje na osnovi trenutnega stanja terminalnega vezaj in zadnjega sprejetega signala. Pri tem se odvisno od vrste zveze v eno celoto lahko poveže tudi po več terminalnih vezij. Poleg stanja in zadnjega odkritega signala (dogodka) vplivajo na zvezo tudi podatki, ki so v pomnilniku procesorja vpisani kot polstalni podatki (SPD-Semi Permanent Data). Ti opisujejo upravičenost medsebojnega povezovanja terminalnih vezij iste centrale in oštevilčenje (način usmerjanja zvez).

Poleg osnovne dejavnosti obdelave pozivov (call processing) opravlja procesor tudi nekatere postranske dejavnosti, ki omogočajo osebju centrale da zagotavlja delovanje centrale v skladu s predpisi. Te dejavnosti obsegajo spremembe polstalnih podatkov, statistično obdelavo dogajanj v centrali, sprotno testiranje delov centrale in podrobnejše testiranje delov centrale, ki so izključeni iz prometa. Vse te dejavnosti

morajo potekati na tak način, da osnovna dejavnost pri tem ni prizadeta niti v obsegu niti v kvaliteti.

2. MERILA DELOVANJA PROCESORJA

Ker je osnovna naloga telefonske centrale vzpostaviti čimveč zvez v čimkrajšem času ob čimboljši kvaliteti, je osnovno merilo uporabljenega procesorja prometna zmogljivost, ki jo označujemo z BHCA (Busyest Houre Call Attempts - število pozivov v glavni prometni uri) in predstavlja število začetih pozivov, ki jih centrala zmore brez redukcije kvalitete delovanja. V praksi je od lastnosti samega procesorja (procesne zmogljivosti) in organizacije programske opreme odvisna zgornja meja tega parametra, od programske opreme same pa, kaj se zgodi, ko je predpisana meja presežena. Idealna rešitev je zavračanje vseh novih pozivov brez kakršnihkoli motenj na ostalih zvezah. sprejemljiva redukcija funkcij je tudi blokada dodatnih aktivnosti (detaljni testi, manj pomembne statistične obdelave). Prav zaradi povečanja vrednosti BHCA so v mnogih centralah procesorja dodatni različni avtomati, ki razbremenjujejo sam procesor pri najbolj rutinskih opravilih (izvajanje daljših operacij, filtriranje motenj, stalne kontrole delovanja in podobno.

Drug zelo pomemben faktor je odzivni čas, ki ga programirani procesor potrebuje, da v normalnem obratovanju ukrepa na osnovi sprejeta signala. Ta čas je podan s signalizacijo in znaša v najhujšem primeru 7 ms (CCITT N°4). Zaradi različnih signalizacij je obdelava na večini terminalnih vezjih lahko počasnejša.

Tretji faktor je kapaciteta delovnega pomnilnika. Ta parameter je težko ocenjevati absolutno, saj poleg same kapacitete, izražene v razpoložljivih bitih, močno vpliva izbrani programski jezik. Večina SPC central ima poleg delovnega še zunanji masovni pomnilnik za obnovitev programov in podatkov v primeru napake in za redkeje uporabljene testne in statistične programe.

3. ISKRINE SPC TELEFONSKE CENTRALE IN NJIHOVI PROCESORJI

Doba SPC telefonskih central se je v Iskri pričela z nakupom licence za Metaconto 10C. Razen izvedenk Metaconte so bile vse naslednje SPC centrale plod lastnega razvoja. To velja v veliki meri tudi za procesorje, ki so rezultat skupnega dela znanstveno raziskovalnih institucij in Iskre.

3.1. METACONTA 10C

Metaconta 10C je bila razvita v ITT kot SPC centrala s hermetičnim spojiščem, namenjena za nivo lokalnih in medmestnih central. Za krmiljenje lokalnih central je uporabljen računalnik 16000, ki ima 16-bitno organizacijo, delovni pomnilnik je feritni, zunanji pomnilnik pa magnetni trak. Za podporo so v centrali enote, ki avtonomno izvedejo časovno daljše akcije (krmiljenje relejev terminalnih vezij, markiranje), sama terminalna vezja pa vsebujejo filtre za izločitev večine motenj.Instrukcijski set procesorja ima nekaj instrukcij, prirejenih posebej za uporabo v telefonskih centralah.

Medkrajevne centrale M10C upravlja 32-bitni procesor 3200, ki je prilagojen za krmiljenje telefonskih central podobno kot 1600, vendar pa je bistveno zmogljivejši. Feritni pomnilnik so Iskrini razvijalci zamenjali s polprevodniškim leta 1982.

3.2. METACONTA 10CN

Tehnološki napredek je pogojeval modernizacijo lokalnih central tipa M10C. Rezultat je centrala M10C, ki ima poleg sprememb v telefonskem delu tudi nov procesor 1602. Ta ima že polprevodniški pomnilnik in večjo hitrost delovanja. Zunanji pomnilnik je magnetni boben, za zamenjavo programov pa služi magnetni trak.

3.3. ZASEBNE CENTRALE E300, E16

Prvi lastni prodor Iskre v področje SPC telefonskih central predstavlja zasebna centrala

tip centrale	kapaciteta ako. vodov	tip procesorja	organizacija procesorja	kapaciteta delovnega pomnilnika	zunanji pomnilnik	hitrost 10^6 ins/m	programski jezik	zmogljivost BHCA	obseg izdelanih programov	število izdelanih procesorjev	opombe
lokalna M10C	30000	Z-1600	16 bitna	128 K feritni	magnetni trak	0,4	zbirnik	70 000 (2 procesorja)	120K	300	posebne instrukcije
medmestna M10C	256000	2 do 6 3200	32 bitna	512 K feritni	magnetni trak	0,55	zbirnik	180000 (2procesorja)	640K	60	posebne instrukcije
M10CR	60000	Z-1602	16 bitna	256K polprevodniški	boben, magnetni trak	0,5	zbirnik tabele	100000 (2procesorja)	450K	120	posebne instrukcije
E15, E300	E15 15 E300 300	programski avtomat *	8bitna	6 K polprevodniški	nima	0,1	zbirnik	92000 **	15K	1750	
E100, E32	E100 128 E32 32	TK 6800	8bitna	64K polprevodniški	nima	0,3	zbirnik	15000	70K	650	
I300R, I500A	1200	RM128 *	8pogojno 12bitna	16K programi 128×32podatki 16K SPD polprevodniški	nima	0,1	zbirnik	90000 **	50K	320	128navideznih procesorjev
I2000	3000	do 35 TK 6800	8bitna	154K polprevodniški	magnetni trak	0,3	zbirnik SL 1	14000 po procesorju	200K	50	distribuirano krmiljenje

* označba procesor je pogojna

** omejitev v arhitekturi centrale

E 300, ki je bila razvita v sodelovanju med Fakulteto za elektrotehniko, Ljubljana in Iskro. Krmilni organ je programski avtomat, posebej prilagojen za vzporedno krmiljenje večjega števila procesov. Avtomat je grajen iz standardnih DTL in TTL vezij.

3.4. I300R, I500A

Primernost rešitve krmiljenja telefonske centrale malih kapacitet s programskim avtomatom in težave pri programiranju in kontroli so pogojevale dodelavo le tega v procesor RM128 (leta 1979). Ta procesor je zanimiv zaradi svoje zasnove, ki že vsebujejo osnovne elemente, potrebne za vzporedno krmiljenje več procesov. Vsebuje namreč 128 navideznih procesorjev, od katerih ima vsak svoj programski števec, set delovnih registrov in delovni pomnilnik. Vsem procesorjem je skupen ROM programski in SPD pomnilnik. Vsak od procesorjev izvrši v enem obhodu le eno instrukcijo, tako da res lahko govorimo o vzporednem in istočasnem odvijanju vseh 128 procesov. Uporabljeni programski jezik je zbirnik RM 128, ki pa ima zaradi skupine podpornih avtomatov težo višjega programskega jezika.

3.5. CENTRALE Z DISTRIBUIRANIM KRMILJENJEM

Družina central I2000 je krmiljena z doma razvitim mikroračunalnikom, zasnovanim na Motorolinem M6800. To je prva Iskrina telefonska centrala, ki ima distribuirano krmiljenje. En mikroračunalnik upravlja do 128 terminalnih vezij majhne zahtevnosti, do 32 terminalnih vezij velike zahtevnosti ali en sklop centraliziranih organov centrale (skupinsko stikalo, administrativni modul). Programski jezik je v prvotnih izvedbah centrale (E100, E32) samo zbirnik, v kasnejših pa kombinacija zbirnika in SL1 (specifikation language 1), razvit v sodelovanjih med Iskro in Institutom Jožef Stefan. Polnjenje in obnavljanje programa poteka iz centralnega administrativnega modula.

4. KAKO NAPREJ

Kot je razvidno iz pregleda, so tehnološke osnove uporabljenih procesorjev izpred leta 1977, kar nedvoumno pomeni, da v celoti ne izkoriščamo možnosti, ki jih je tehnološki napredek ponudil v tem času. Razvijalci Iskre ta zaostanek nadomeščajo po eni strani z modernizacijo obstoječih procesorjev (uvedba polprevodniškega pomnilnika namesto feritnega, uvedba višje integracije v TK6800) po drugi strani pa se pripravlja za uporabo v distribuirano krmiljenih sistemih 16-bitni računalnik na osnovi M68000. Sprejemljiv nivo cen tudi za pomnilnike istočasno omogoča tudi uvajanje višjih programskih jezikov. Iskra bo zato tudi v distribuirano krmiljenih centralah prešla na 16-bitne procesorje in višje programske jezike povsod tam, kjer bodo omogočale tržne razmere.

Reference: interna dokumentacija
Iskre Telematike, Kranj

NOVA GENERACIJA SPEKTROFOTOMETARA

P.Hinić, Elektrotehnički fakultet Banjaluka
Z.Majkić, SOUR "RUDI ČAJAVEC" Banjaluka
S.Talić, M.Hinić, Regionalni medicinski centar Banjaluka

UDK: 681.3.06

SADRŽAJ - U ovom radu prikazan je novi koncept projektovanja spektrofotometara za UV i vidljivo i infracrveno područje. Sa potpunom zastupljenošću mikroprocesorske tehnologije, ovaj savremeno projektovani spektrofotometar sa jednim mlazom, daje povećanu tačnost i pojednostavljene operacije: automatska kompenzacija struje mraka, podešavanje nule itd.

THE NEW GENERATION OF SPECTROPHOTOMETERS - This paper deals with new concept in terms of simplicity and design, to the field of UV, infrared and visible spectrophotometry. By fully utilising microprocessor technology this completely new design single beam spectrophotometer gives increased instrument accuracy combined with simplified operation: automatic dark current compensation, zero setting and so on.

1. UVOD

Spektrofotometri su složeni elektroničko-optički sistemi namijenjeni za mjerenje hemijskog sastava ispitivane materije. Široka im je primjena u medicini, biologiji, naftnoj industriji. Njihov razvoj kretao se od jednostavnih uredjaja do složenih procesnih računara i uglavnom je zavisio od razvoja tehnologije. Danas su spektrofotometri dostigli takav stepen razvoja, da je to potpuno automatizovani mjerni sistem, sa svojim periferijama, koji se može priključiti na računarski sistem i vršiti složenu hemijsku analizu. Predloženi spektrofotometar realizovan je u mikroprocesorskoj tehnici i spada u klasu najsavremenijih uredjaja koji se danas pojavljuju na tržištu.

2. PRINCIP RADA

Svjetlosni zrak emitovan iz izvora svijetlosti je obično širokopojasni fenomen sastavljen od više spektralnih komponenata, dolazi na optički sistem gdje se razlaže na monohromatsku svjetlost. Optički sistem je jednostavan, ako ga čine samo optički filtri koji propustaju svjetlost uskog spektralnog pojasa. Medjutim, ako se želi svjetlost veoma uskog spektralnog opsega ili monohromatska, osnova optičkog sistema je monohromator (optička prizma), precizni i složeni dio optičkog sistema.

Monohromatska svjetlost odredjene talasne dužine prolazi kroz ispitivani uzorak. Samo odredjena komponenta u materiji apsorbuje datu monohromatsku svjetlost koja pada na senzor, pretvarač svjetlosti u električnu veličinu. Ovdje počiva osnova ove metode, gdje se iz iznosa električne veličine odredjuje stepen apsorbcije odnosno sastav materije. Sa senzora električna veličina se vodovodi na A/D konvertor, pa na složeni mikroračunarski sklop, koji vrši zahtijevanu obradu izmjerene veličine. U ovom radu biće dat kratak opis realizacije ovoga sklopa.

3. REALIZACIJA I PROGRAMSKA PODRŠKA MIKRORAČUNARA

U ovom radu ne mogu biti opisana električna kola i podsklopovi, koja čine hardversku strukturu mikroračunara, već će biti opisan mikroračunar kao integralna sklopovska cjelina. Zatim ćemo izložiti postupak mjerenja, uzimanje, obrada i prikazivanje rezultata. Na kraju, veoma kratko biće riječi o programskoj podršci mikroračunara.

3.1. Prikaz sklopovske realizacije mikroračunara

Na priloženoj slici data je hardverska struktura mikroračunarskog sistema. Osnovni sklop je mikroprocesor uP 8085. Memorijski dio sastoji se od radne memorije (RAM) 3x6116 i programske memorije (EPROM) 3x2716. Za radnu memoriju koristi se i pripadna memorija u ulazno-izlaznoj jedinici 8155. Ove jedinice takodje služe kao prihvatni registri i kola za povezivanje DATA BUS-a mikroprocesora sa analogno-digitalnim (A/D) i digitalno-analognim (D/A) konvertorima i ostalim periferijama mikroračunara.

Za pretvaranje paralelnog u serijski kod i obratno namijenjen je univerzalni sinhrono-asinhroni sklop 8251.

Osnovna komponenta koja povezuje mikroprocesor, tastaturu i pokazivač (display) je 8279.

Sklop 8212 prihvata donji adresni bajt i sa gornjim adresnim dijelom čini 16-bitni adresni bus.

Dekoderi 8205 služe za odabiranje pojedinih sklopova u toku rada mikroračunara (čip selekt) i prelaz sa 3-bitne na 8-bitnu riječ.

8216 je kolo za formiranje upravljačkih signala (bafer-drajver).

3.2. Transfer podataka

Sa senzora analogni signal dovodi se na ulaz programabilnog pojačala. Pojačanjem ovog pojačala upravlja mi-

kroprocesor. Poslije analogno-digitalne konverzije, rezultati mjerenja, preko ulazno-izlazne jedinice 8155 do vode se u radnu memoriju 6116.

Nakon aritmetičkih operacija, rezultati mjerenja izraženi preko transmisije, ekstincije ili koncentracije vode se preko sklopa 8279 na pokazivač. Prenos ovih podataka sa DATA BUS-a vodi se na kolo 8251, gdje se vrši pretvorba paralelnog u serijski kod. Isto tako, preko jednog kola 8155 vodi se 8-bitna ili 16-bitna riječ na sklopove koji prihvataju informaciju u paralelnom kodu, dok se preko drugog kola 8155 i digitalno-analognog konvertora rezultati mjerenja iskazuju u analognom obliku.

Program koji vrši obradu i prenos podataka smješten je u EPROM-u 1x2716.

Da bi sagledali rad mikroračunarskog sistema i obavljanje zadane funkcije, konkretna problematika može se podijeliti u četiri dijela:

1. Podešavanje pojačanja i mjerenje I_0.
2. Mjerenje I, traženje odnosa $T=I/I_0$ i prikazivanje T na displeju.
3. Izračunavanje logaritma $E= - \log T$ i prikazivanje E na displeju.
4. Izračunavanje količnika K=E/C i prikazivanje K na displeju.

3.2.1. Podešavanje pojačanja i mjerenje I_0

Podešavanje pojačanja se vrši kad izvor analognog signala daje najveću (referentnu) vrijednost, i to sve dotle, dok se na izlazu konvertora ne pojavi maksimalna vrijednost koju on može da da. To je vrijednost I_0. Razlog za ovo je jasan: promjena pojačanja (a time i digitalne vrijednosti I_0) ne utiče na odnos $T=I/I_0$ i ostale parametre signala, ali veća vrijednost I_0 omogućava bolju diskretizaciju analognog signala ($0 < I \leq I_0$), tj. istom području promjene analognog signala odgovara više diskretnih vrijednosti ako je I_0 veće.

Analogni signal male vrijednosti se prvo pojača, va zatim se vrši njegova A/D konverzija. Klok konvertora i start impuls obezbjedjuju brojačke sekcije kola 8155. br.1 i 8155 br.2., respektivno. Potprogram START obezbjedjuje start impulsa, nakon završene konverzije, ponovno uspostavljanje START ulaza na visok nivo (što, istovremeno, uzrokuje obaranje linije koja označava kraj konverzije, na nizaknivo) i ispitivanje uslova potrebnih za pravilno učitavanje podataka.

Pošto kontrolna riječ pojačanja ima osam bita, njena promjena za jedan bit izaziva promjenu digitalnog izlaza konvertora za 2^4. To znači da jednoj vrijednosti pojačanja odgovara 16 različitih digitalnih vrijednosti na izlaz konvertora (pri istom analognom ulazu). Pčema tome, 4 najniža bita su slučajni brojevi i njihova vrijednost se ne ispituje kad se vrši podešavanje pojačanja.

3.2.2. Mjerenje nove vrijednosti analognog signala (I), traženje odnosa $T=I/I_0$ i prikazivanje vrijednosti za T.

Učitavanje nove vrijednosti analognog signala se vrši potprogramom ULAZ. On poziva potprograme CLOCK i START čime obezbjedjuje potrebne uslove za rad konvertora.

Treba napomenuti da će izlaz konvertora biti različit od nule čak i onda kad ne postoji analogni signal koji se ispituje. Zbog toga, učitane vrijednosti za I_0 i I nisu stvarne vrijednosti analognog signala, već im je superponiran neki parazitni signal I_p. Zapravo, učitane su vrijednosti:

$$I_0' = I_0 + I_p$$
$$I' = I + I_p$$

Zato se, prije traženja odnosa T, vrši oduzimanje vrijednosti parazitnog signala I_p od učitanih vrijednosti ulaznog signala I_0 i I, tj.

$$T = \frac{I}{I_0} = \frac{I - I_p}{I - I_p}$$

(potprogram PRIP).

Vrijednost parazitnog signala se dobije učitavanjem digitalne vrijednosti sa izlaza konvertora, pri čemu je ulazni signal jednak nuli. Ovo je potrebno učiniti uvijek nakon promjene uslova rada kola, a prije prvog mjerenja vrijednosti za I.

U svim programima brojevi su predstavljeni sa predznakom (u obliku dvojnog komplementa) i pokretnim zarezom, koristeći, pri tome, za sve aritmetičko-logičke operacije parove registara, tj. oblik svih brojeva je

$$z_d d_{14} d_{13} \ldots d_1 d_0 \; 2^{z_e e_6 \ldots e_0}$$

3.2.3. Izračunavanje logaritma E=-logT i količnika K=E/C

Približno izračunavanje vrijednosti E=-logT se vrši razvojem u Taylor-ov red logaritamske funkcije. Kada se odredi ekstikcija E, konstanta C se unosi preko tastature, a nakon toga izračunava se koncentracija K=E/C.

Rezultati se prikazuju u dekadnom brojnom sistemu na displeju. Ovi se rezultati preko sklopa 8251 vode na periferijske jedinice, koje zahtijevaju informaciju u digitalnom obliku, a preko D/A konvertora u analogne mjerne jedinice.

4. ZAKLJUČAK

Istraživanja i razvoj na ovom zadatku trajala su tri godine. Rad se odvijao u više faza. Osnovni cilj bio je doći do savremenog fotospektrometra, namijenjenog na biološke i industrijske laboratorije. Ovo je zahtijevalo izučiti mjerne metode i predložiti nove koje se koriste

na današnjem stepenu tehnološkog razvoja.

Rezimirajući zaključke istraživanja može se zaključiti slijedeće:

1. Usavršena je mjerna metoda, kod koje se koristi isti svjetlosni put, odnosno isti detektor, za referentnu i mjernu komponentu. Kod ove metode i kod svih sličnih, osnovni problem je bio podešavanje nulte vrijednosti instrumenta i referentne ekstinkcije, od čega je uglavnom zavisila tačnost rezultata mjerenja. Ovo je otežavalo i rukovanje sa mjernim sistemom, na kojem je bilo ugradjeno više potenciometara, za podešavanje. Podešavanje nulte vrijednosti pojedinih sklopova (pojačavača, detektora i konvertora) i referentne ekstinkcije riješeno je softverski. Pri svakom ciklusu mjerenja provjerava se i podešava nulta vrijednost instrumenta i referenca ekstinkcije, tako da rezultati mjerenja ne zavise od temperaturnih i drugih uslova.

2. Izabran je širokopojasni detektor koji obezbjedjuje upotrebu instrumenta u kliničkim i industrijskim laboratorijama.

3. Ugradjeno je programabilno pojačalo tako da se referentna ekstinkcija može dovesti na traženu vrijednost. Pojačanjem pojačala upravlja mikroprocesor.

4. Razvijen je elektronski mikroračunar čija je osnovna komponenta mikroprocesor uP 8085. Obezbijedjeni su izlazi i ulazi za analogni i digitalni signal, za paralelni i serijski kod. Rezultati mjerenja pokazuju se na led-diodnom pokazivaču. Komuniciranje sa uredjajem vrši se preko heksadecimalne tastature.

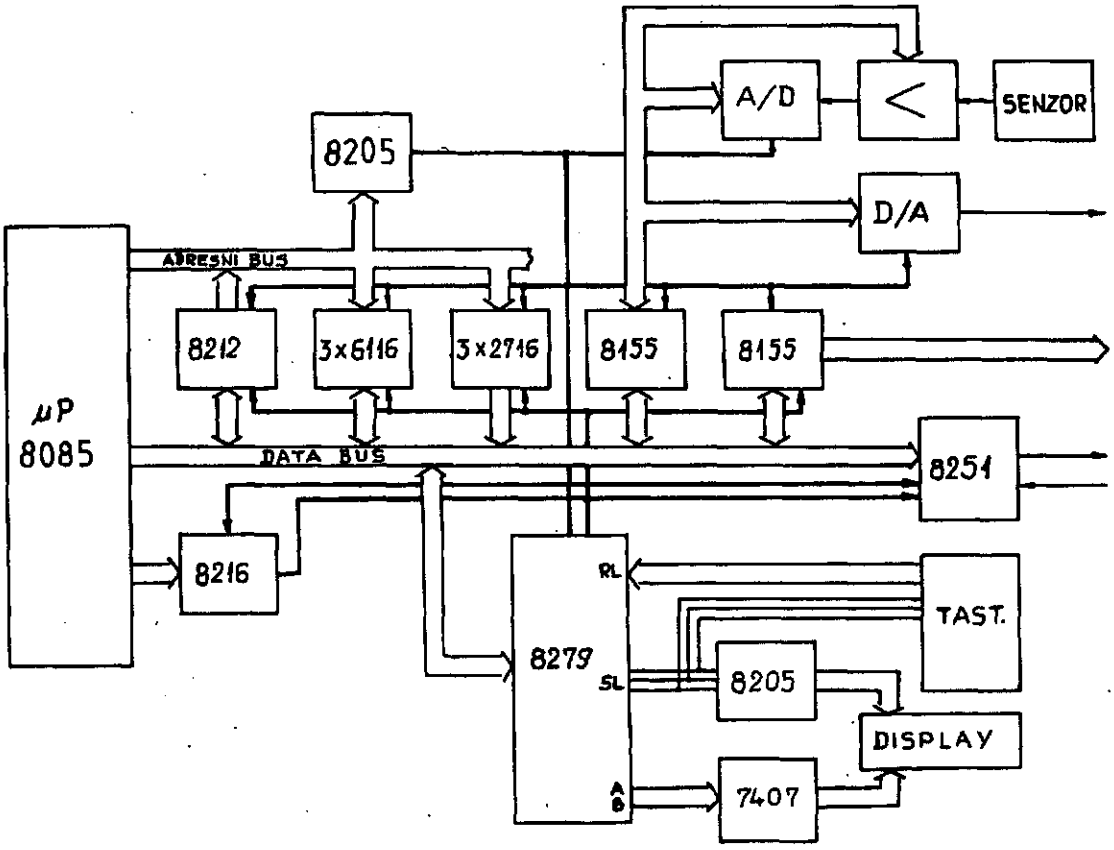
Izbor mjerne veličine ostvaruje se preko tri tipke: transmisija, ekstincija i koncentracija. Računske operacije obavljaju se u aritmetici tekućeg zareza.

Softver mikroračunara obezbjedjuje komuniciranje preko ulazno-izlaznih vrata sa periferijama mikroračunara i izračunavanje pomenutih mjernih veličina. Kompletna sklopovska realizacija izvedena je na tri štampane ploče.

5. LITERATURA

1. Gema A. Streitmatter Hito Flore Microprocessors Theory and Applications A Praktice-Hall company.

2. Christopher A. Titus TEA: An 8080/8085 CO-Resident Editor/Assembler, Howard W.Sams. & CO., Inc.



MIKRORAČUNALNIŠKO VODENI TL ANALIZATOR

M.Mihelič, U.Miklavžič, Z.Rupnik, P.Satalić

Institut "Jožef Stefan", Univerza Edvarda Kardelja,
Ljubljana, Jugoslavija

UDK: 681.3.06

POVZETEK: V prispevku so nanizane osnovne zahteve upoštevane pri razvoju termoluminescenčnega analizatorja namenjenega tako rutinskemu določanju doz, kot laboratorijskemu raziskovalnemu delu ter podane rešitve z mikroračunalnikom, ki so privedle do uspešne realizacije naprave.

SUMMARY: In the paper the principal requirements followed in the design of the thermoluminescent analyzer intended for the routine dose readings, as well as for the laboratory research work are described, and an outline of the microcomputer-based solutions which led to successful realization of the device is given.

UVOD

Prodor mikroračunalniške tehnologije posredno povzroča, da tudi v dozimetriji, zlasti v osebni - kjer se obdeluje veliko število dozimetrov, novejše računalniško prilagodljive fizikalne metode dobivajo dodatno težo in izpodrivajo klasične načine merjenja. Taka sodobna metoda je termoluminescenca, ki sloni na lastnosti določenih dielektričnih materialov, da po radiaktivnem obsevu oddajo pri segrevanju del absorbirane energije v obliki vidne svetlobe. Množina izsevane svetlobe je zato zelo linearno merilo prejete doze. Dozimetri, izdelani iz takih materialov imajo več prednosti pred klasičnimi: s svojim linearnim svetlobnim odzivom pokrivajo izjemno široko področje doz (od 10^{-6} Gy do 10^{4} Gy), prenesejo ekstremne dozne hitrosti, so ponovno uporabni in so zaradi svoje majhnosti tudi nadvse priročni. Vendar se navzlic svojim izjemnim lastnostim TL dozimetrija v rutinski rabi še ni popolnoma udomačila. Vzroke gre iskati v razmeroma zahtevnem čitanju in interpretaciji rezultatov ter v različnih karakteristikah posameznih vrst materialov. To zahteva dodatno znanje in ustrezno aparaturno opremo.

Sodobne mikroračunalniške metode danes omogočajo elegantno reševanje problemov vodenja merilnih procesov, obdelave rezultatov in njihovega prikaza. Rezultat uporabe takih metod je večnamenski mikroračunalniški termoluminescentni analizator MR-200, ki je bil razvit na IJS in je namenjen tako rutinski dozimetriji kot raziskovalni rabi. Namen tega prispevka je prikazati osnovne poteze mikroračunalniških metod uporabljenih pri koncipiranju in realizaciji merilnika.

MERILNI PROCES

Lastnost termoluminescenčnih materialov, da si "zapomnijo" sprejeto dozo, je mogoče izkoristiti na več načinov. Postopek, ki je glede na izvedbo morda manj zahteven, je tak, da dozimetrsko tableto, ki je bila predhodno izpostavljena radioaktivnemu sevanju, segrevamo na električnem grelcu po predpisanem programu in merimo oddano svetlobno energijo. Pri tem je časovni potek jakosti izsevane svetlobe (žarilna krivulja) odvisen od časovnega poteka segrevanja. Iz poteka žarilne krivulje in ob upoštevanju lastnosti uporabljenega TL materiala, je mogoče računsko določiti prejeto dozo.

Za realizacijo merilnika, ki bi izpolnjeval zahteve merilnega procesa, zagotavljal reproducibilnost meritev in s tem dajal uporabne rezultate ter hkrati omogočal ovrednotenje izmerjenih vrednosti in prikaz rezultatov, je potrebno izvesti (slika 1):

- Programirano segrevanje dozimetrske tablete. Ker je časovni potek jakosti izsevane svetlobe, s tem pa tudi evaluacija svetlobnega signala, odvisen od časovnega poteka temperature dozimetrske tablete, je potrebno zagotoviti dovolj dobro ponovljivost temperaturnega poteka. Hkrati je zaradi različnih lastnosti posameznih TL materialov potrebno omogočiti izbor različnih časovnih potekov segrevanja (n.pr. predogrevne faze, maksimalne temperature itd.).
- Odjemanje svetlobnega signala in pretvarjanje v napetost v izjemno širokem področju linearnih svetlobnih odzivov TL materialov (9 dekad). Uporaba fotopomnoževalke kot senzorja daje dobre rezultate, vendar le pri

različnih delovnih režimih in dovolj dobri stabilizaciji le-teh. To zahteva visokostabilni visokonapetostni izvor za napajanje, napetostne preklope in termostatiranje same fotopomnoževalke.

- Kalibriranje odjetih vrednosti z referenčnim svetlobnim virom, kar naj bi zagotavljalo neodvisnost rezultatov od morebitnih sprememb režimov delovanja elementov v merilni verigi.
- Upravljanje s celotno merilno aparaturno opremo, ki sega od priprave meritve (izbor parametrov in režimov meritve), sprotnega (on-line) vodenja samega merilnega procesa, do obdelave odjetih vrednosti, njihovega ovrednotenja po izbrani metodi ter prikaza rezultatov (slika na monitorju, izpis na printerju).

REALIZACIJA MERILNIKA Z MIKRORAČUNALNIKOM

Pokazalo se je, da lahko mikroračunalnik s svojimi zmogljivostmi pri odjemanju in oddajanju analognih vrednosti, shranjevanju digitalnih podatkov, računski obdelavi in prikazu rezultatov, zadovoljivo opravlja vlogo krmilnega elementa v merilniku in izpolnjuje vse naštete zahteve. Pri izdelavi naprave smo se naslonili na doma razviti mikroračunalniški sistem. Zgrajen je modularno z vsemi potrebnimi vmesniškimi funkcijami (ADC, DAC, paralelni/serijski prenos, paralelni vhodi/izhodi, videoterminal). Preko sistemskih modulov je mogoče priključiti zunanje pomnilniške enote (digitalna kaseta, gibki disk), kar omogoča delo z večjimi količinami podatkov. Te lastnosti dajejo sistemu veliko fleksibilnost in možnost učinkovitega prilagajanja potrebam načrtovalca. V sledečih odstavkih so opisane osnovne poteze rešitev z mikroračunalnikom, ki smo jih vgradili v merilni sistem za termoluminescenčno dozimetrijo izdelan na IJS (slika 2).

Regulacija temperature grelca TL tablet

Zaprtozančni temperaturni regulator grelca je izveden docela programsko. Mikroračunalnik preko A/D pretvornika v konstantnih časovnih intervalih odjema napetost termočlena. Merilni konec termočlena je pritrjen na grelec, referenčni pa na termostatirano ohišje fotopomnoževalke. Na podlagi znane temperaturno-napetostne odvisnosti uporabljanega termočlena, mikroračunalnik sproti izračunava trenutno absolutno temperaturo grelca in shranjuje temperaturne vzorce v pomnilnik. Iz predpisanega časovnega temperaturnega poteka in odjetih temperaturnih vzorcev s pomočjo regulacijskega algoritma sproti izračunava vrednosti krmilnih signalov. Preko D/A pretvornika in močnostne stopnje z njimi krmili moč gretja in s tem temperaturo grelca (slika 3).

Odjem svetlobnega signala

Mikroračúnalnik odjema vrednosti vzorcev jakosti izsevane svetlobe v fiksnih časovnih intervalih preko 12 bitnega A/D pretvornika z vhodnim območjem 0 - 10 V. Da bi lahko izrabili celotno izjemno široko odzivno področje TL materialov, ki segajo od normalnih doz v okolju do akcidentnih, smo razvili tokovno-napetostni pretvornik z nastavljivim ojačanjem in visokonapetostni vir za napajanje fotopomnoževalke z nastavljivo izhodno napetostjo. Oba nastavljiva parametra sta pod nadzorom mikroračunalnika. Vsakemu merilnemu rangu, ki je določen z ojačanjem I/U pretvornika in napetostjo na fotopomnoževalki, pripada koeficient s katerim mikroračunalnik normira napetost, odjeto preko A/D pretvornika v tem rangu. Tako določi izmerjenemu svetlobnemu vzorcu absolutno vrednost v skali celotnega merilnega področja instrumenta. Preklapljanje ojačanja pretvornika in visoke napetosti omogoča odjemanje svetlobnih signalov v željeno širokem področju, ki ustreza dozam od 5.10^{-7} Gy do 5 Gy (7 dekad).

Sprotna kalibracija merilnika

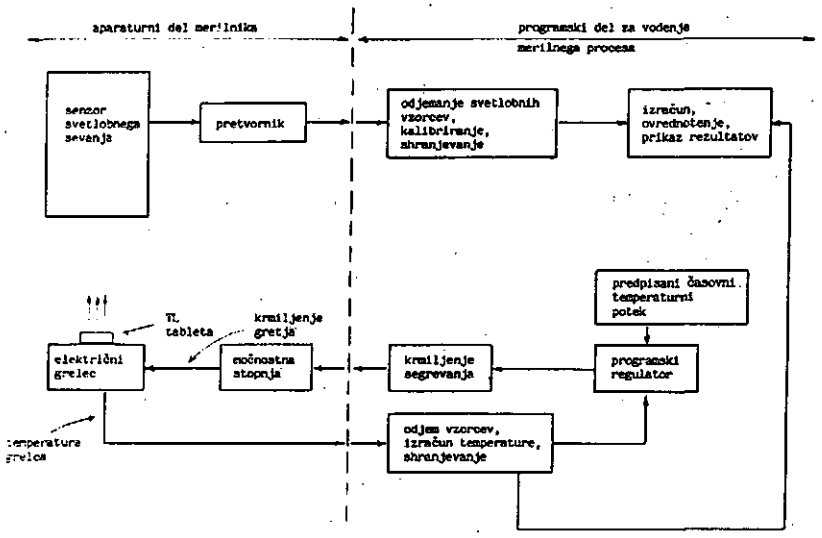
Uporaba mikroračunalnika pri vodenju merilnega procesa omogoča odpravo vplivov sprememb ojačanja in ničle (driftov) v merilni verigi na rezultate meritev. Z uporabo takoimenovane sprotne digitalne kalibracije je mogoče občutno izboljšati karakteristike merilnika brez uvajanja visokokvalitetnih elektronskih komponent.

Bistvo te metode je v tem, da računalnik pred vsakim merilnim procesom izmeri dve referenčni točki, ki nato služita za kalibracijo vsakega odjetega vzorca (slika 4). Pri tem točnost meritve ni več odvisna od časovne stabilnosti posameznih elementov v merilniku (fotopomnoževalka, I/U pretvornik, A/D pretvornik), ampak je popolnoma naslonjena na točnost in stabilnost referenčnega svetlobnega vira. Zato je njegovi izdelavi posvečena posebna pozornost. Realiziran je kot precizno termostatirana LE dioda (patentirano).

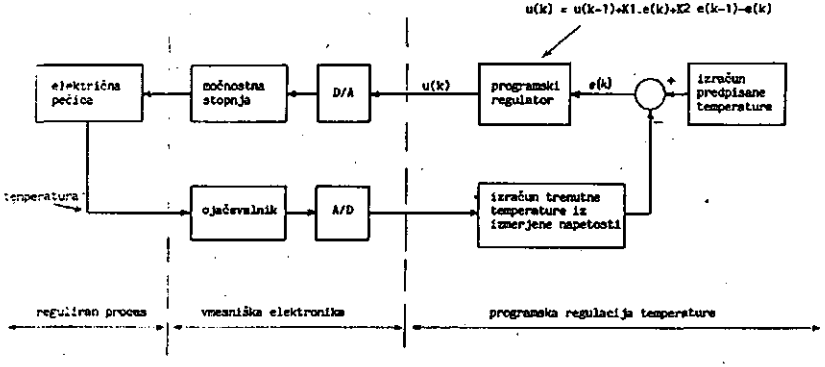
Upravljanje merilnika

Mikroračunalnik preko vgrajene programske opreme vodi celoten proces od izbora željenih parametrov meritve, same meritve, do interpretacije in prikaza rezultatov. Posebej za to izdelan operacijski sistem povezuje posamezne programske enote in skrbi za komunikacijo med operaterjem in aparaturo preko tastature in monitorja. Sestavljajo ga trije nadzorni programi, ki se javljajo vsak preko svojega prikaza na monitorju (slika 5).

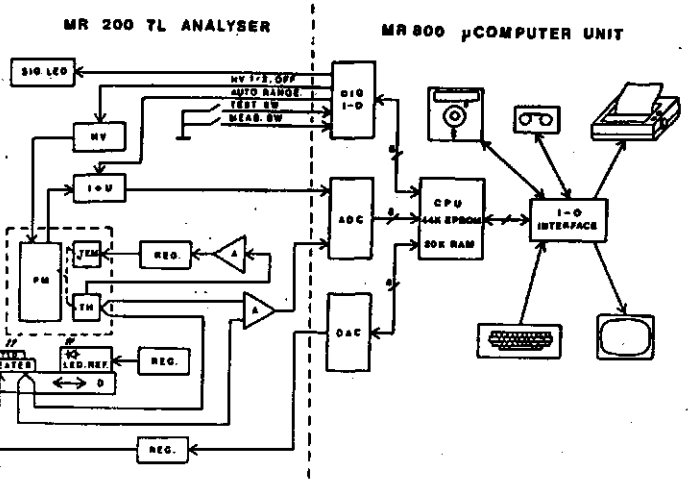
Nadzorni program TLD je osnovni program s pomočjo katerega je mogoč dostop do vseh funkcij sistema. Z njegovimi ukazi je mogoče po želji definirati merilne parametre in startati meritev, iz podatkov v pomnilniku po končani meritvi izrisati graf žarilne krivulje in krivulje poteka temperature, sprožiti izračun doze po izbranem evaluacijskem programu in preslikati sliko z ekrana na papir v tiskalniku. Poleg tega vsebuje nadzorni program TLD ukaze za delo s kartotekami, kar omogoča sistemu izvajanje knjigovodskih funkcij v primeru rutinskega dela z večjo količino dozimetrov. Iz tega pro-



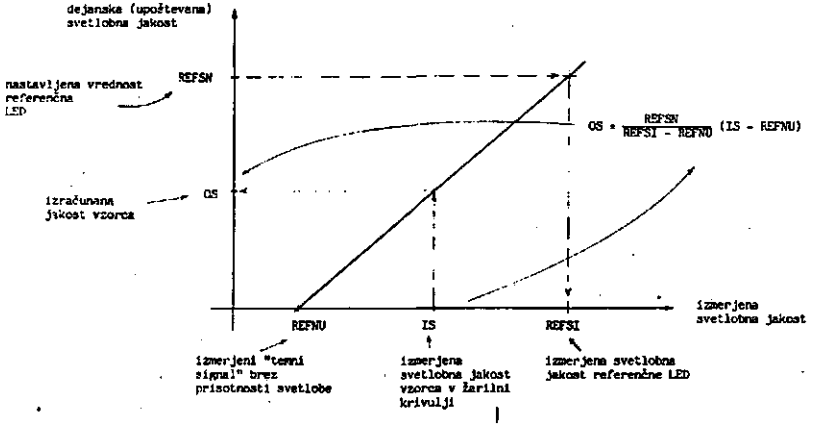
Slika 1: Koncept TLD merilnika



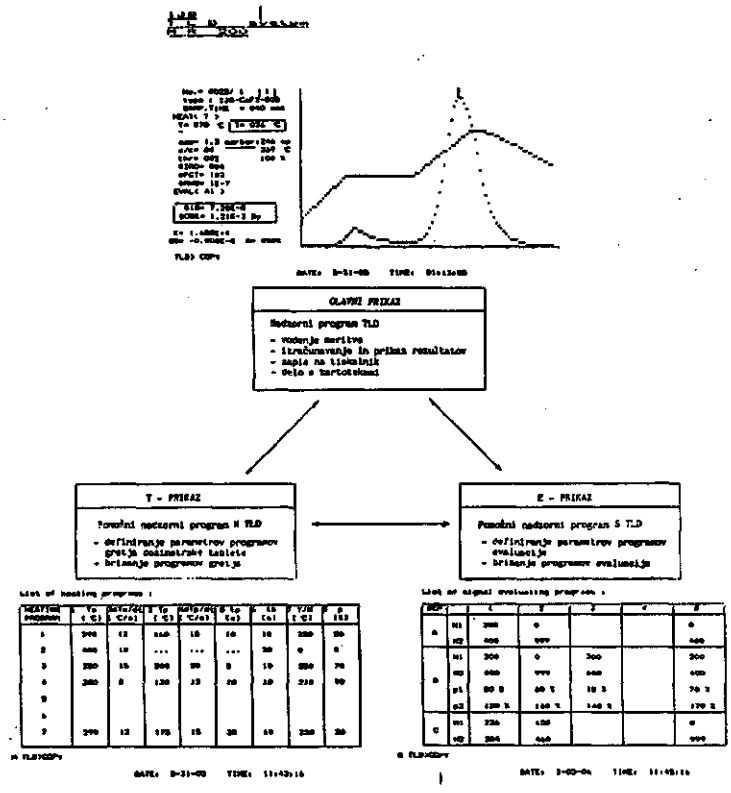
Slika 3: Shema regulacije temperature pečice



Slika 2: Blokovna shema termoluminiscentnega merilnika MR-800



Slika 4: Princip sprotne kalibracije izmerjenih vzorcev



Slika 5: Shema operacijskega sistema

grama je z ukazom mogoč prehod na oba pomožna nadzorna programa.

Pomožni nadzorni program STLD deluje preko E-prikaza, ki kaže tabelo evaluacijskih programov, po katerih računalnik lahko ovrednoti žarilno krivuljo. Z ukazi pomožnega nadzornega programa STLD je mogoče po želji definirati parametre petnajstih evaluacijskih programov. Preko glavnega prikaza jih lahko nato operater vključuje v obdelavo rezultatov različnih ali ene in iste meritve.

Pomožni nadzorni program HTLD deluje preko T-prikaza, ki prikazuje tabelo temperaturnih programov, po katerih računalnik med meritvijo krmili segrevanje dozimetrske tablete. Parametri temperaturnega programa, ki jih je mogoče nastaviti s pomočjo tastature, določajo časovni potek temperature merjene tablete. Z ukazi nadzornega programa je možno definirati do sedem grelnih programov.

ZAKLJUČEK

Lastnosti, zmogljivosti in možnosti mikroračunalnika pri snovanju in realizaciji opisanega procesno-merilnega sistema so se izkazale kot bistvene. Le z računalniško zmogljivo enoto je bilo mogoče v napravi združiti procesno vodenje merilnega postopka s sprotnim kalibriranjem izmerjenih vzorcev, računsko obdelavo merjenih vrednosti in njihov prikaz. Komunikacija med napravo in uporabnikom preko terminala je učinkovita in preprosta, saj so z ukazi operacijskega sistema hitro dosegljive vse funkcije merilnika. Hkrati daje uporabniku možnost vpliva na široko paleto parametrov meritve, obdelave in ovrednotenja rezultatov ter njihovega prikaza. Zasnova programske opreme daje napravi lastnosti, ki omogočajo tako rutinsko delo brez potrebe po globjem poznavanju principov TL dozimetrije, kot laboratorijsko raziskovalno delo s TL materiali, zlasti z možnostjo analize žarilnih krivulj s pomočjo gibljivega markerja in nastavljanja različnih ključnih parametrov procesa. Brez dvoma bi bile s klasičnimi (neračunalniškimi) pristopi k izgradnji merilnika take karakteristike nedosegljive.

Večmesečno testiranje MR-200 TL analizatorja je pokazalo upravičenost zastavljenega koncepta in potrdilo pričakovanja. Seveda pa se s tem odpirajo nove zahteve, ki jih zasnova naprave podpira. Ena prvih je morda avtomatska menjava merjenih dozimetrov in čitanje imen nosilcev preko identifikacijskih števil vkodiranih na ohišjih dozimetrov (črtasta koda). Delo na teh področjih nas čaka v naslednjem obdobju.

MIKRORAČUNALNIŠKI SISTEM ZA KONTROLO PROCESA VULKANIZACIJE

Saša Prešern
Institut Jožef Štefan
Ljubljana, Jugoslavija

UDK: 681.3.014

Z uvedbo računalniškega vodenja posameznega sektorja industrijskega procesa lahko zagotovimo boljšo kvaliteto izdelka in omogočimo racionalnejšo rabo surovin. Kritičen sektor proizvodnje v gumarski industriji je kontrola delovanja preš. Kvaliteta izdelka zavisi od stopnje vulkanizacije, ki je določena glede na fiksne vhodne parametre. Ker pa v procesu proizvodnje prihaja do nepredvidenega nihanja vhodnih parametrov (tu imamo v mislih tlak in temperaturo), pa izdelek ni vulkaniziran vedno pod takimi pogoji kot je bilo predvideno. S sprotnim spremljanjem temperature in tlaka v preši in ob podpori mikroračunalnika lahko v vsaki preši sproti računamo stopnjo vulkanizacije, ki je funkcija temperature in izdelek ob pravem času odstranimo iz preše ter dosežemo pravo stopnjo vulkanizacije vsakega izdelka.

MICROCOMPUTER SYSTEM FOR CONTROL OF VULCANISATION PROCESS With microcomputer control of certain industrial preocess one can asure beter quality of products and optimal usege of material. A critical sector in industrial process oki tyres is on line control of temperature and pressure. Quality of each product depends on degree of vulcanisation which is a function of temperature and time. Because temperature is not fixed value but slightly fluctuates up and down, the vulcanisation effect is not always as expected. With on line control of temperature and presure and with microcomputer control of vulcanisation effect the industrial production is under control all the time for every product.

1. UVOD

Problemi, ki se pojavljajo pri procesnem vodenju v gumarski industriji so splošne narave in jih v takšni ali drugačni obliki srečamo v večini industrijskih procesov (REM79, HOL83, PREŠ5A). Te problemi so:

- kako definirati projekt
- kako izbrati in zasnovati strojno računalniško opremo,
- kako doseči modularnost programske opreme in hkratno izvajanje več procesov v realnem času na istem mikroračunalniku,
- kako zmanjšati vpliv motenj,
- kako organizirati komunikacijo računalnika z operaterjem, da bo ta čim enostavnejša
- kako zbirati in obdelovati podatke o industrijski proizvodnji in s tem omogočiti pregled nad učinkovitostjo dela.

V tem članku obravnavamo nekatere najzanimivejše probleme te vrste, s katerimi smo se srečali pri uvajanju računalniškega vodenja delovanja preš v gumarski industriji (PREŠ5B).

Računalnik za vodenje procesa vulkanizacije mora izvajati temperaturno in tlačno spremljanje procesa vulkanizacije ter sprotno analitično spremljanje stopnje vulkanizacije za izdelavo potniških in tovornih plaščev. Računalnik, ki vodi delovanje posamezne preše, sprejema signale iz te preše ter spremlja stanje kalike. V primeru, da je tlak ali temperatura kalike izven predpisane vrednosti, se preša ne zapre. Delovanje vseh preš naj bo zaradi enakomerne obremenitve kotlarne čim bolj enakomerno razporejeno. Z doseganjem optimalnega časa vulkanizacije, ki ga sproti računa mikroračunalnik za vsako preso, dosežemo boljšo kvaliteto izdelka, optimalno kapaciteto strojev in s tem vplivamo na stroške tehnološkega procesa vulkanizacije. Računalniško vodenje naj obsega tudi dnevne statistične obdelave proizvodnega procesa po prešah, delavcih in proizvodih kar omogoča hiter povratni vpliv na boljšanje kvalitete prizvodnje.

Postopek vulkanizacije obdelamo tako, da v podrobnosti razdelamo sam tehnološki postopek, nakar zgradimo matematični model, na podlagi katerega mikroracunalnik računa stopnjo vulkanizacije.

Odvisnost hitrosti reakcije od temperature podaja Arrhenijeva enačba (WOL67):

$$RG(2) = RG(1)\, e^{-\frac{A'}{R}\left(\frac{1}{T2}-\frac{1}{T1}\right)}$$

kjer je

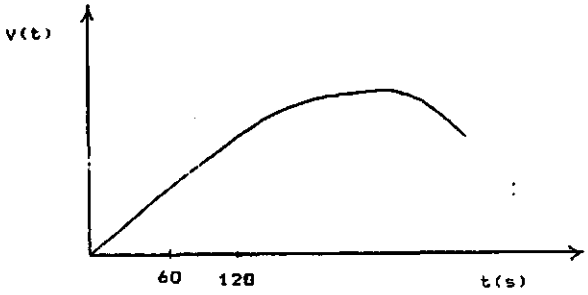
RG = hitrost reakcije
R = plinska konstanta (1,986 * 10exp-3 kcal/Mol)
T = temperatura v stopinjah kelvina
A' = aktivacijska energija

Vulkanizacijski efekt V podaja število molekul, na katerih je reakcija že potekla in je sorazmeren času poteka reakcije.

$$V(t) = \int_0^t e^{-\frac{A}{T(t)}}\, dt$$

Stopnjo vulkanizacije Vo smo določili v laboratoriju in je poznana vrednost. Integral v zgornji enačbi pomeni vulkanizacijski efekt v

preši, kjer se temperatura nasičene pare T(t) s časom spreminja. Približni čas vulkanizacije v preši določimo tako, da poznamo potek temperature v plašču na testnih merilnih mestih in računamo integral V v odvisnosti od časa. Ko je stopnja vulkanizacije V enaka stopnji predpisane vulkanizacije Vo, je postopek segrevanja končan in preša se odpre.



Slika 1.: Spreminjanje stopnje vulkanizacije s časom.

2. STROJNA RAČUNALNIŠKA OPREMA

Pri določitvi strojne računalniške opreme imamo na voljo več različnih prijemov:

- zbirni mikroračunalniki
- centralni računalnik
- distribuirani sistem

Vsak pristop ima svoje prednosti in slabosti. Pozorno je treba pretehtati značilnosti posameznih pristopov, ker s tem vplivamo na stroške celotne investicije in tudi na učinkovitost sistema. Na odločitev o izbiri sistema vplivajo:

- število vhodnih in izhodnih signalov,
- potreba po komunikaciji med stroji (prešami),
- potreba po neodvisnosti vsakega stroja (preše),
- operacijski sistem,
- programska oprema za razvoj aplikativnih programov,
- programska oprema za izvajanje aplikativne programske opreme po korakih in testiranje

Na vsaki preši imamo 6 vhodnih signalov v računalnik (analognih) in 4 izhodne signale (digitalne). Preše morajo delovati vsklajeno. Za to zadostuje, da za vsako preso vodimo v računalniku evidenco o njenem trenutnem stanju, ki je zaprta, odprta ali v okvari. Iz vsake prese dobimo v računalnik signal o stanju prese. Razvoj programske opreme mora potekati na standardnem operacijskem sistemu. Programska oprema za razvoj aplikativnih programov mora vsebovati urejevalnik teksta, zbirnik in prevajalnik za pascal. Programska oprema za izvajanje aplikativne programske opreme po korakih in testiranje mora vsebovati "debugger" in testne postopke za testiranje vseh vnodnih in izhodnih signalov.

Pri analizi vseh potrebnih lastnosti računalniš kega sistema za vodenje pres se izkaže, da najbolje ustreza zahtevam industrijskega procesa sistem z zbirnimi mikroračunalniki, kjer vsak mikroračunalnik nadzira 20 pres. S tem pristopom dosežemo sledeče:

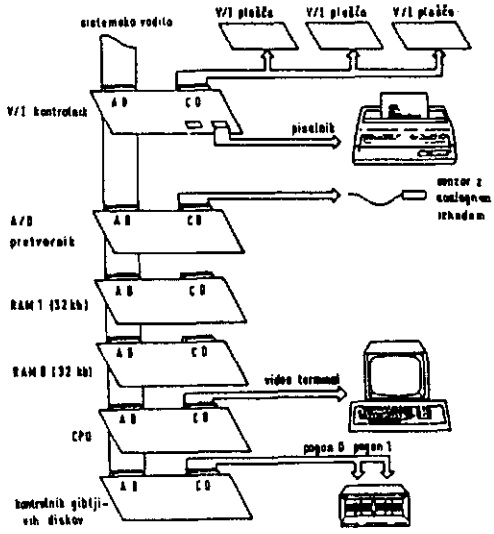
- optimalno izkoristimo kapacitete posameznega mikroračunalnika in dosežemo najboljše razmerje pri ceni računalniškega sistema na preso, saj se izognemo velikemu številu posameznih vezij in povezav, ki bi bilo potrebno pri popolnoma porazdeljenem sistemu.

- omogočimo vodenje 20 pres nekrati brez potrebe po komunikaciji med procesorji, ki bi bila potrebna v primeru distribuiranega sistema,

- izognemo se zahtevi po dodatni programski in strojni računalniški opremi (vmesniki, tastature, prikazovalniki), ki je potrebna pri porazdeljenih sistemih,

- izognemo se dodatnemu delu operatrja (vpisovanje kode, vpisovanje mejnih vrednosti temperature in tlaka) na vsaki preši, ki bi bilo potrebno pri porazdeljenih sistemih,

- v primeru okvare računalnika imamo spremembo iz računalniškega vodenja na avtomatski režim dela samo pri skupini 20 pres ne pa v celotnem proizvodnem procesu, kar bi se zgodilo, če bi imeli en centralni računalnik.



Slika 2.: Konfiguracija strojne opreme mikroračunalnika za vodenje sistema preš.

Skupno sprejema računalnik za vodenje sistema 20 pres 40 digitalnih vhodnih signalov in oddaja 80 digitalnih izhodnih signalov. Če uporabimo vhodne plošče velikosti enojna evropa za sprejem 8 signalov in izhodne plošče za oddajo 16 signalov potrebujemo po 5 plošč vsake vrste.

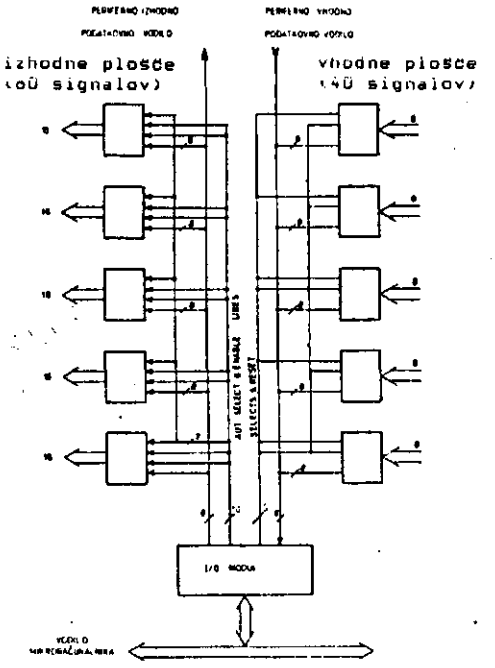
3. PROGRAMKSA OPREMA

Pri določitvi programske opreme izhajamo iz dejstva da:

- uporabimo čim več obstoječe programske opreme (sistemska programska oprema, aritmetic ni paketi, multiprogramsko jedro itd),

- s pomočjo programskih pripomočkov čim hitreje realiziramo aplikacijo.

Aplikativna programska oprema mora omogočati izvajanje programa v realnem času. Analiza je pokazala, da je zadostna frekvenca ponavljanja programa enkrat na sekundo. Če izberemo 8 bitni računalnik, mora biti večina programske opreme za sprotno spremljanje in vodenje industrijskega procesa pisane v zbirnem jeziku. Če pa se odločimo za 16 bitni računalnik, je



Slika 3.: Blok shema organizacije vhodno izhodnih linij.

večji del programske opreme realiziran v višjem programskem jeziku.

Aplikativna programska oprema vsebuje za vsako preso, ki je priključena na mikroračunalnik naslednje rutine:

- INICIALIZACIJA - nastavitev:
  * šifre delavca
  * šifre plašča
  * vrednosti za čas vulkanizacije
  * aktivacijske energije
- TEMPERATURA - čitanje temperature
- TIPKE - čitanje tipk stop, start na preši
- SINHRONIZACIJA - sinhronizacija vključevanja preš
- delovne rutine za različne faze
  * ČAKA - čakanje na sinhronizacijo
  * VKLOP - start vulkanizacije
  * VULKANIZIRA - izračun stopnje vulkanizacije
  * IZKLOP - odpiranje preše
- LUČKE - signalizacija (zelena, rdeča, rumena)
- KOMANDE - ukazi računalnika preši
- ČAS - sinhronizacija časa

Računalnik vsebuje še vrsto tabel, to so:

- tabela za stanje vseh preš,
- tabelo šifer delavcev,
- tabelo limitnih vrednosti temperature,
- tabelo limitnih vrednosti tlaka,
- transformacijsko tabelo za pretvorbo signalov iz senzorja temperature v stopinje celzija,
- tarnsformacijsko tabelo za pretvorbo signalov iz senzorja tlaka v bare,
- tabelo vseh možnih napak,
- tabelo sporočil za ukrepanje.

Aplikativna programska oprema je napisana tako, da se je je možno naučiti uporabljati v nekaj urah. Vsa komunikacija med operaterjem in računalnikom je v obliki menujev in vprašalnikov.

Statistična obdelava se izvaja enkrat dnevno. količine, ki jih računalnik zbira in obdeluje so naslednje:

- skupna dnevna proizvodnja plaščev po vrstah plaščev. Za vsako vrsto plašča se izpiše šifre plaščev (5 mestno število) in količina.

- za vsako preso se izpiše: šifra delavca, število proizvedenih plaščev pod to šifro za vsak kalup (na vsaki preši sta dva kalupa), vrsta izjemnih situacij in njih skupno število (izjemne situacije so: ročno odpiranje preše, odpiranje preše, ker je predviden čas vulkanizacije potekel, čeprav izračunana stopnja vulkanizacije še ni dosežena).

- za tovorne plašče se izpišejo zaporedne številke plaščev, preša, šifra delavca, datum in čas vulkanizacije

- izračuna se skupni čas delovanja preše in skupni čas, ko je preša odprta.

Statistične obdelave se izpisujejo enkrat dnevno in arhivirajo v sami proizvodnji saj to omogoča sprotno kontrolo nad stanjem proizvodnje, stanjem preš, vrsti napak, ki se občasno pojavljajo, storilnistjo delavcev in tehnološkimi pogoji pare, ki vplivajo na proizvodnjo.

Po 30 uran se obdelani statistični podatki avtomatično izbrišejo.

4. KOMUNIKACIJA Z OPERATERJEM

Komunikacija med računalnikom in operaterjem je izvedena v obliki menujev in odgovorov na vprašanja.

Ob vklopu računalnika se pojavi na zaslonu glavni menu. Operater mora vtipkati svojo šifro, šifo plašča in čas vulkanizacije ter laboratorijsko temperaturo in čas. Če je šifra plašča enaka kot pri delu, ki ga je opravil operater v prejšnji izmeni, lahko odtipka operater v tej izmeni kar tipko return. Enako velja za čas vulkanizacije ter za laboratorijsko temperaturo in čas. Nato se mu na zaslonu prikaže slika, ki kaže trenutno stanje celotnega proizvodnega procesa za sistem 20 preš (Slika 4.)

```
SISTEM ZA VODENJE PROCESA VULKANIZACIJE
=======================================

Ö - SPREMEMBA ŠIFRE OPERATERJA
Š - SPREMEMBA ŠIFRE PLAŠČA
Č - SPREMEMBA ČASA VULKANIZACIJE
P - SPREMEMBA STANJA POSAMEZNIH PREŠ

TEMPERATURA PRENIZKA ZA: P15, P16
POTEKEL ČAS VULKANIZACIJE ZA PREŠO: P5
SINHRONIZACIJA VKLJUČEVANJA PREŠE: P9

-----------------------------------------
STANJE PREŠ:
ZASEDENE:  P1, P4, P5, P6, P7, P10, P11,  P13,
           P15, P16, P17, P18, P20
ČAKANJE/Š: P9
ČAKANJE/T: P15, P16
PROSTE:    P2, P3, P8, P14
V OKVARI:  P3, P12, P19
```

Slika 4.: Signalizacija na zaslonu med delovanjem linije.

na zaslonu se prikazuje stanje preš in vsa sporočila v zvezi z napakami, čakanjem in temperaturami.

Statistične obdelave se izpisujejo enkrat dnevno na tiskalnik. Vse statistične izpise izvaja samo za to pooblaščena oseba in ne operater proizvodnje na presah. Primer izpisa statistike proizvodnje po presah kaže slika 5.

STATISTIKA PROIZVODNJE PO PREŠAH 14.1.1985

PREŠA	ŠIFRA DELAVCA	ŠT. PLAŠČEV KAL A	KAL B
P1	00127	23	21
	00214	28	29
	00156	12	26
P2	...		
.			
.			
.			
P20	00127	23	20

Slika 5.: Izpis na tiskalnik za statistiko dnevne proizvodnje po presah.

5. ZAKLJUČEK

Program uvajanja računalnika v vodenje preš poteka postopno. Celoten program traja 18 mesecev, s tem, da je ena preša vodena z računalnikob že po 8 mesecih.

Pred prenosom računalniškega vodenja sistema preš v industrijsko okolje se izdela laboratorijski preizkus. Izdela se simulator za delo ene preše in se preizkusi delovanje računalnika in programske opreme na simulatorju. Delo na simulatorju se demonstrira uporabnikom, ti pa podajo morebitne pripombe. V nadaljevanju se izdela tudi simulator za preizkus dela celin preš na enem računalniku.

Podali smo primer uvajanja računalniškega vodenja industrijskega procesa. Problemi, ki smo jih srečali pri našem delu so podobni tudi pri drugih projektih uvajanja računalnika v proizvodnjo. Velcek izkušenj iz projektov uvajanja mikroračunalnikov v proizvodnjo posredujemo v tem sestavku. Vodilo tega in podobnih projektov pa je dej- stvo, da računalnik omogoča boljšo kvaliteto proizvodnje, izključi človeški faktor, ki je lahko vzrok za napake ter omogoča racionalnejši pregled nad zalogami in statistiko proizvodnje.

6. LITERATURA

(HOL83) R.C. Holland: Microcomputers for process control, Pergamon Press, 1983.

(PREŠ5A) Š. Preširn: Mikroračunalnik v spremljanju in vodenju industrijskih procesov, Zavod za tehnično izobraževanje Ljubljana, Ljubljana, 1985.

(PREŠ56) Š. Preširn: Senzorsko tipanje in vodenje procesa vulkanizacije, Sava, interni dokument, 1985.

(REM79) U. Rembold: Prozess uno mikrorechner systeme, R. Oldenbourg Verlag GmbH, Munchen, 1979.

(WOL67) D.I. Wolf: Handbuch fur Vulkanisationszeitbestimmung, Semperit - Sava, interno poročilo, 1967.

METEOROLOŠKA AVTOMATSKA MERILNA POSTAJA AMP - STOLP ZA DOLOČEVANJE DISPERZIJE V ATMOSFERI PRI NE KRŠKO

B.Diallo, B.Glavič, M.Lesjak, P.Mlakar, Z.Polak

Institut "Jožef Stefan", Univerza Edvarda Kardelja,
Ljubljana, Jugoslavija

UDK: 681.3.014

POVZETEK: V referatu je opisana meteorološka avtomatska merilna postaja AMP-Stolp, ki meri meteorološke parametre za določevanje disperzije v zraku. Opisano je mesto AMP-Stolp v ekološkem informacijskem sistemu NEK, zgradba postaje in problematika določevanja razredčevanja v ozračju.

ABSTRACT: In the paper we describe the automatic weather station AMP-Stolp which is a part of the ecological information system NE Krško. We discuss about the method of dispersion modeling in the air from point sources and data requirements needed to characterize airborne dispersion.

Avtomatska merilna postaja AMP-Stolp je del ekološkega informacijskega sistema (1), ki zbira in obdeluje meteorološke podatke in podatke o koncentraciji radioaktivnih snovi v okolici Nuklearne elektrarne Krško (NEK). Ekološki informacijski sistem je sestavljen (slika 1) iz koncentratorja, štirih merilnih postaj in uporabnikov, ki so povezani s koncentratorjem preko UKV, telefonske ali stalne žične zveze. Glavne naloge ekološkega informacijskega sistema so:

- zbiranje meteoroloških parametrov iz neposredne bližine in okolice Nuklearne elektrarne (NE) po priporočilih ameriške agencije za zaščito okolja za kompleksen model razredčevanja,
- zbiranje podatkov o izpustu radioaktivnih plinov in delcev iz NE,
- zbiranje podatkov o hitrosti doze iz okolice NE,
- prenos vseh parametrov v večji računalnik, kjer se nadaljnje obdelajo in arhivirajo,
- prenos meteoroloških in radioloških podatkov v TPC (tehnični podporni center NEK), v center Elektrogospodarstva Slovenije in Hrvaške,
- prikaz vseh podatkov za potrebe NE,
- ocenitev doz v smeri širjenja oblaka po različnih metodah.

Osrednji del informacijskega sistema je koncentrator, ki poleg komunikacije skrbi še za delovanje celotnega sistema, merjenje izpusta radioaktivnosti žlahtnih plinov, izotopov joda in zračnih partikulatov, računanja razredčitvenih koeficientov, prikaza vseh podatkov ter ocenitev doz okoliškega prebivalstva v smeri širjenja radioaktivnega oblaka v primeru nesreče.

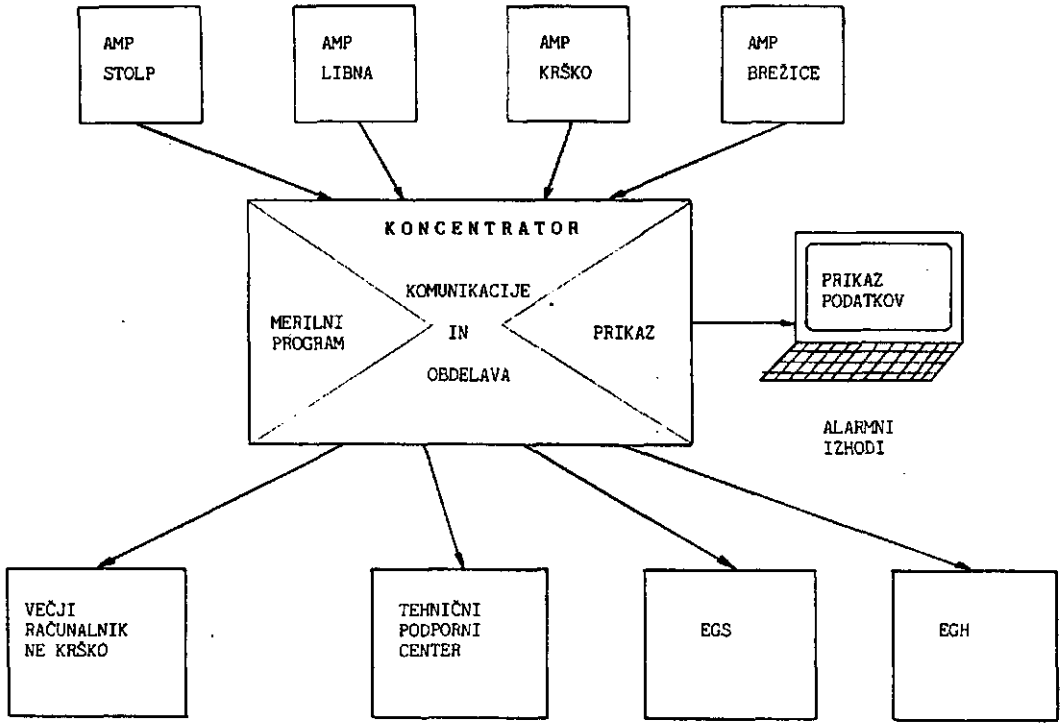
Avtomatske merilne postaje zbirajo podatke o meteoroloških in radioloških parametrih: hitrost in smer vetra, temperatura zraka, relativna vlaga, zračni pritisk, sončna radiacija, količina padavine in koncentracija radiaktivnih snovi. Iz zbranih podatkov se računajo poprečne polurne in dnevne vrednosti, ekstremne vrednosti in standardne deviacije. Izvedeni podatki se redno zapisujejo na digitalno magnetno kaseto in se na zahtevo pošljejo v koncentrator.

AMP-Stolp ima pomembno vlogo v ekološkem informacijskem sistemu NEK, ker mora zagotoviti koncentratorju vse potrebne meteorološke podatke za čimboljšo oceno razredčitvenih koeficientov. Za razliko od ostalih treh merilnih postaj, AMP-Stolp zbira meteorološke podatke na štirih različnih nivojih: 70 m, 40 m, 10 m in 2 m. Podatki, ki jih AMP-Stolp pošlje koncentratorju, so vektorska hitrost in smer vetra, skalarna hitrost vetra, ekstremne hitrosti vetra in pripadajoče smeri, standardne deviacije x in y komponente hitrosti vetra, količina padavin in poprečne ter ekstremne vrednosti in standardne deviacije za ostale meteorološke parametre.

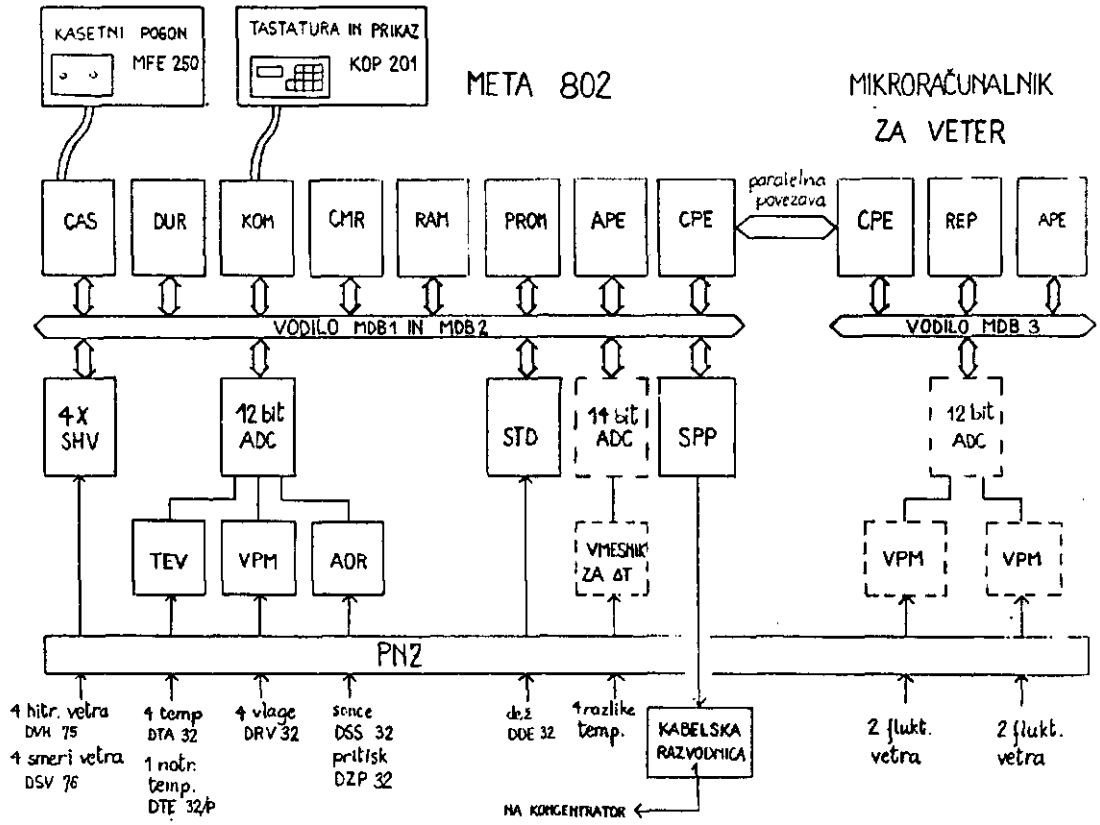
AMP-Stolp je sestavljena iz dveh mikroračunalnikov: (3) METE 802 in mikroračunalnika za veter. Mikroračunalnika sta sestavljena iz standardnih modulov, ki so logično zaključene celote (slika 2).

METO 802 sestavljajo:

CPE - centralna procesna enota: 8 bitni procesor 8080, 1 k RAM, 6 k EPROM in V/I enota,
APE - aritmetična procesna enota: pospeši računanje v plavajoči vejici,
PROM - 20 k EPROM pomnilnika,



Slika 1: Ekološki informacijski sistem NEK



Slika 2: Zgradba AMP-Stolp

RAM - 8 k RAM pomnilnika,
CMR - 2 k CMOS RAM pomnilnika z avtonomnim napajanjem,
KOM - vmesnik za tastaturo in prikaz na čelni plošči postaje,
DUR - ura realnega časa s štiriletnim koledarjem,
CAS - vmesnik za magnetno kasetno enoto za arhiviranje podatkov,
SPP - serijsko paralelni prenos: serijski prenos podatkov v koncentrator (20 mA zanka),
SHV - digitalni vmesnik za smer in hitrost vetra,
STD - digitalni vmesnik za dež,
ADC - A/D pretvornik in 16-kanalni multiplekser,
AOR - analogni vmesnik za sonce in pritisk,
TEV - analogni vmesnik za temperaturo,
VPM - analogni vmesnik za vlago.

Predviden je še vmesnik za razliko temperature in 14-bitni A/D pretvornik.

Pri mikroračunalniku za veter je realiziran računalniški del, ker senzorji za fluktuacije še niso razviti. Povezava med mikroračunalnikoma je realizirana preko modulov CPE, na katerih je V/I enota.

Programska oprema je za vse merilne postaje enaka, ne glede na število meteoroloških parametrov, ki jih obravnava postaja. To omogoča prilagodljiva zasnova programske opreme, kjer se delovanje in konfiguracija postaje določita s tabelami. Vsak meteorološki parameter, ki ga meri in obdeluje merilna postaja predstavlja logični parameter z določeno kodo, ki je lahko vklopljen ali izklopljen. Vsi vklopljeni parametri se nahajajo v tabeli vklopljenih parametrov. Poleg te tabele imamo še glavno in prikazovalno tabelo, ki vsebujeta podatke o načinu odjema, obdelavi, zapisu in prikazu podatkov.

Mikroračunalnik za veter je podrejen delovanju METE 802, ki ga krmili s prekinitveno linijo in paralelnim vodilom. S tem je dosežena sinhronizacija mikroračunalnika za veter z delovanjem METE 802. Računalnika sta povezana s paralelnim vodilom, preko katerega META 802 usmerja delovanje mikroračunalnika za veter in od njega sprejema podatke. META 802 pošlje zahtevo za povprečenje podatkov in za nov obdelovalni cikel. Mikroračunalnik za veter povpreči podatke in jih odda METI. Nato ob vsaki prekinitvi vzorči parametre vetra in tvori delne vsote, dokler ni zahteve za ponoven obdelovalni cikel in se vse ponovi. Če META 802 zahteva trenutne podatke, jih odda. Oblika podatkov, ki se prenašajo, je standardna in je enaka za zahtevo, kakor tudi za odgovor.

SOH	N	K1	K2	VSEBINA SPOROČILA	CRC	EOT

Slika 3: Oblika sporočila med METO in MR za veter. SOH in EOT sta sinhronizacijska ASCII znaka, N je število vseh podatkov, K1 koda pošiljatelja, K2 je koda naslovljenca in CRC je ostanek deljenja pri ciklični zaščitni kodi.

Komunikacija med METO 802 in mikroračunalnikom za veter je izvedena preko V/I enot, ki sta na CPE-jih. Jedro V/I enote je standardni vmesnik i8255. Vmesnika sta direktno povezana s "flat" kablom, tako da je aparaturna oprema minimalna. Prenos podatkov je paralelen in asinhronski. Hitrost prenosa je do 100 k bitov/s. Komunikacijo vodi META 802, mikroračunalnik za veter pa je popolnoma podrejen.

NE predstavlja za okolico stalno nevarnost, ki pa jo je potrebno zmanjšati na minimum. Obstaja nevarnost, da pride do okvare na elektrarni in s tem do nezgodnega izpusta. Radioaktivni oblak se širi glede na obstoječe vremenske razmere. Da bi pravočasno evakuirali ogroženo prebivalstvo, je potrebno zasledovati in napovedovati širjenje oblaka. Širjenje in koncentracija plinov v oblaku je odvisno od razredčevalnih sposobnosti atmosfere. Zato je potrebno zagotoviti ustrezne meteorološke podatke in podatke o izpustu, jih obdelati in pravočasno posredovati uporabnikom.

Najpomembnejši parameter za oceno razredčevanja je stabilnost atmosfere. Ko temperatura pada z višino $1^{o}C$ na 100 m, je atmosfera na meji med stabilno in nestabilno (Pasquill-ov razred D). Če temperatura hitreje pada, postane atmosfera bolj nestabilna (Pasquill-ovi razredi A,B,C A je oznaka za najbolj nestabilno ozračje). Če pa temperatura počasneje pada kot $1^{o}C$ na 100 m ali pa celo narašča z višino, je atmosfera stabilna (razredi E, F, G G najbolj stabilna atmosfera).

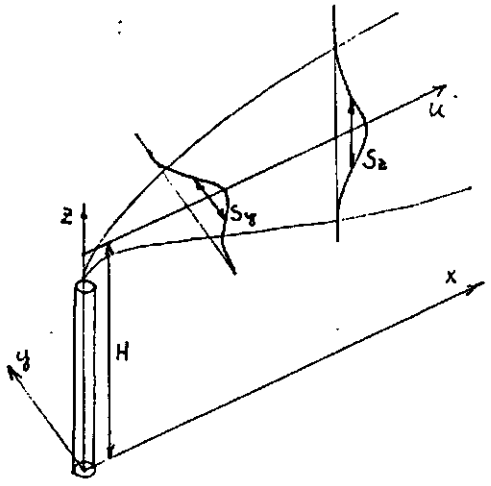
Vertikalni temperaturni gradient povzroča termične turbolence. Čim bolj je atmosfera nestabilna, tem močnejše so turbolence in mešanje zraka. Dodatno termični turbolenci se pojavlja tudi mehanska turbolenca, predvsem zaradi valovite površine zemlje in ovir na njej. To vpliva na vertikalni in horizontalni profil vetra, ki nam otežita oceno razredčevanja. Za oceno disperzije je pomembna tudi višina mešanja zraka (za večje oddaljenosti od izvora), ki je običajno med 100 m in nekaj 1000 m nad površino, in hitrost vetra. Ko veter narašča, se plini oziroma majhni delci razredčijo v večjem volumnu. Tako lahko razredčevanje v atmosferi ločimo na transport, ki je pogojen s povprečnim zračnim tokom, in na difuzijo, ki je pogojena z naključnim gibanjem zraka. Posebno pomembno predvsem za naše razmere pa je meandriranje oblaka. Oblak ne potuje v ravni črti, temveč dela meandre, kot počasi tekoča reka na ravnini. To se dogaja pri šibkih vetrovih (6 m/s) in nevtralni (D) ter nestabilni (E,F,G) atmosferi. Posledica meandriranja je povečano razredčevanje in upočasnitev oblaka.

V zadnjih desetletjih se je ocenitev atmosferske disperzije spreminjala zaradi eksperimentalnih rezultatov in posebno računalnikov, ki zmorejo izračunati kompleksnejše modele.

Najosnovnejši model razredčevanja je enoslojni Gauss-ov model, ki ga je razvil Pasquill (1961) in modificiral Gifford (2). Splošna enačba je:

$$X(x,y,z) = Q/(2*3.14*Sy*Sz*u)*\exp(-0.5*(y/Sy)**2)* *(\exp(-0.5*(z-H/Sz)**2)+\exp(-0.5*(z+H/Sz)**2)$$

X = koncentracija plina v zraku
x,y,z = koordinate kot so definirane na sliki 4,
Q = hitrost izpusta,
Sy,Sz = standardni deviaciji razredčevanja v smeri Y in z,
H = višina izpusta
u = povprečna hitrost vetra



Slika 4: Razširjanje oblaka plina po predpostavkah Gauss-ovega modela.

Standardni deviaciji sta odvisni od turbolentnosti (stabilnosti) atmosfere, višine nad površino zemlje, razgibanosti terena kvalitete površine (travniki, gozd ...), oddaljenosti od izvora in časa povprečevanja. Sy in Sz podajajo avtorji modela kot funkcije stabilnosti in razdalje v obliki formul in grafično. V literaturi se dobijo različne enačbe in grafi, ki se med seboj razlikujejo. Vzrok temu so eksperimentalni rezultati, preko katerih so avtorji določevali standardne deviacije razredčevanja.

Ocena za hitrost vetra je povprečna hitrost med H-2Sz in H+2Sz. Stabilnejše plasti nad ozračjem, kjer se razredčujejo plini, omejijo mešanje. Zato osnovna enačba razredčevanja velja le do razdalje x, kjer je Sz = 0.47L (L = višina mešanja).

V literaturi so si edini, da je najtežje določiti standardni deviaciji razredčevanja Sy in Sz. Turner podaja v svojem članku točnosti za Sy in Sz. Točnost Sy ocenjuje znotraj faktorja 2 za razdalje nekaj 100 m od izvora, za Sz pa znotraj faktorja 2 za razdalje med 100 m in 1000 m. To velja za naslednje predpostavke:

- kontinuirana emisija izvora ali čas emisije večji kot čas potovanja do določene točke v smeri vetra, tako da je difuzija v smeri vetra zanemarljiva;
- difuzija plina ali aerosola (delci manjši kot 20 um diameter), ki ostanejo daljši čas v zraku;
- ves emitiran material se premika v smeri vetra in ni depozita;
- Gauss-ova porazdelitev v smeri y in z;
- ocenitev točnosti velja za sorazmerno močne vetrove (ni meandriranja) in na odprtem agrarnem terenu.

Gauss-ov model je primer posredne metode izračuna razredčitvenih koeficientov. Poznamo pa tudi neposredne metode, ki so novejše. Razlika med tema dvema vrstama metod je v določevanju turbolenc v zraku. Posredne metode temeljijo na določitvi stabilnosti atmosfere, preko katere izračunamo Sy in Sz - standardni deviaciji razredčevanja prečno na širjenje oblaka in v vertikalni smeri. Te metode so bile razvite v Združenih državah Amerike na podlagi meritev v ravninskih področjih, kjer pihajo stalni močni vetrovi. Zato te metode ne dajejo dovolj dobrih rezultatov pri šibkih vetrovih in stabilnem ozračju na valovitem terenu, kot je to običajno v Krškem. V takih pogojih pride do meandriranja oblaka. Neposredne metode pa temeljijo na meritvah razredčevanja na kraju izpusta. Te metode so zahtevnejše, toda rezultati so točnejši.

Za vse metode izračunavanja razredčitvenih koeficientov je pomembno zanesljivo in točno določevanje stabilnosti atmosfere. Kaj rado se zgodi, da se rezultati razlikujejo za en razred. To se dogaja predvsem pri manjših hitrostih vetra, kar je čest pojav v Krškem. Metod za izračun stabilnosti je več, temeljijo pa na temperaturnem gradientu, hitrosti in fluktuaciji vetra. V prvi fazi, ki je izdelana, se stabilnost določa s pomočjo temperaturnega gradienta in hitrosti vetra na višini 10 m. Temperaturni gradient se določa z razliko temperature na 70 m in 2 m s standardnimi aspiriranimi dajalniki temperature (točnost 0.3°C). Ker je temperaturni gradient običajno zelo majhen (0.1 - 1°C), je vprašljiva točnost meritev. Zato je v razvoju merilnik razlike temperature s točnostjo 0.1°C in ločljivostjo 0.02°C. Tudi merjenje hitrosti in smeri vetra ne ustreza popolnoma za izračun fluktuacij vetra. Zato poteka razvoj posebnih senzorjev za merjenje fluktuacij.

Zaradi neustreznih obstoječih senzorjev za temperaturo in veter ter metode izračunavanja razredčitvenih koeficientov, je prišlo pri izdelavi AMP-Stolp do vprašanja ali se izdelava odloži do trenutka, ko bodo na voljo ustrezni senzorji in metoda, ali pa se izdelava opravi v večih fazah. Odločili smo se za drugo varianto - za postopno izdelavo. Pri tem smo morali predvideti senzorje in vmesnike, ki še niso razviti, zagotoviti mikroračunalnik, ki bo sproti obdelal vse podatke, in programsko opremo, ki bo omogočala spreminjanje in razvoj obdelav.

LITERATURA

1. B.Paradiž, Projektne zasnove, Elektroinstitut Milan Vidmar, februar 1985

2. D.Bruce Turner, Workbook of Atmospheric Dispersion Estimates, Environmental Protection Agency, 1970

3. B.Diallo, M.Lesjak, Prilagodljiva zasnova programske orpeme AMP, Informatica 2/3, 1983

METROLOŠKI INFORMACIONI SISTEM

ARBUTINA-JOVIĆ MIRJANA,
ČRNOVRŠANIN ESAD
V.Z. "ORAO" - RAJLOVAC
SARAJEVO
JUGOSLAVIJA

UDK: 681.3.007

Metrološki informacioni sistem povećava produktivnost u metrološkoj laboratoriji i oslobadja prostor za kreativan rad. On pomaže rukovodstvu prilikom planiranja i praćenja procesa baždarenja, a neposrednim izvršiocima u samom procesu baždarenja. Tehnologija baždarenja je standardizovana i unijeta u računar, te se automatski generišu radni i izvještajni zapisnici. Svi podaci o procesima baždarenja i okruženju se unose u centralizovanu bazu podataka mrežnog tipa (IDS/II). Tako korisnik ima na raspolaganju potpuno integrisane i ažurne mjerne podatke. Svi programi su radjeni u jeziku visokog nivoa - COBOL-u, uz korištenje najmodernijih tehnika struktuiranja. Kao rezultat toga, oni su vrlo čitljivi te je potreban minimalan napor za dorade i održavanje. Paralelno su razvijene i batch i on-line verzija sistema. Sistem je u svakodnevnoj eksploataciji već dvije godine i pokazuje zadovoljavajuće rezultate u radu.

METROLOGICAL INFORMATION SYSTEM

Metrological information system increases productivity in metrological laboratory and makes free space for creative work. It helps management in planning procedures and gaging process control, and it also helps immediate workers in gaging process. Gaging technology is standardized and entered into the computer, so work-protocols and report-protocols are generated automaticly. All data about gaging process and environment are maintained in a centralized data base (network data base IDS/II). This means that user has access to fully integrated up-to-date metric data. All programs are written in a high-level language - COBOL, using the latest structured techniques. That's why these programs are very readable so modification and maintenance efforts are minimized. Batch and on-line version of system are available. System is in everyday-use for two years now and gives satisfactory working results.

1. UVOD

Mjerni sistem se sastoji od niza aktivnosti i mehanizama koji proizvode mjerne informacije potrebne društvu. On je integralni dio proizvodnje, nauke, trgovine i svih naših aktivnosti. Ovaj sistem ima posebno značajno mjesto u avionskoj industriji, gdje su rigorozni zahtjevi u pogledu kvaliteta završnog proizvoda. Neka ispitivanja pokazuju da na mjerno ispitivanje pojedinih proizvoda otpada oko 30% vremena ukupne izrade.

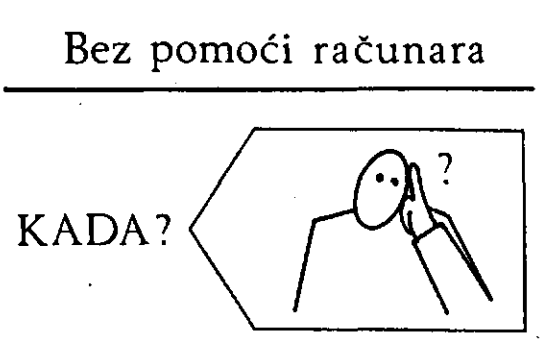
Da bi korištenje mjerne opreme imalo zadovoljavajuće efekte, neophodno je da se ona izbaždari u tačno odredjenim vremenskim intervalima. Ako mjerna oprema nije izbaždarena na vrijeme, može se desiti da rezultat mjernog ispitivanja ne bude korektan, što direktno utiče na smanjenje kvaliteta proizvoda. S druge strane, dešava se da baždarenje predugo traje što uzrokuje zastoje u proizvodnji i, samim tim, smanjenje produktivnosti. Sve ovo utiče na smanjenje konkurentnosti naše proizvodnje na svjetskom tržištu.

Ako potražimo uzroke svih ovih problema, oni su obično u prevelikoj količini podataka koje treba ručno pratiti i obraditi. Kada se govori o prometu velikih količina podataka, onda se odmah nameće misao da najveću pomoć u tome može da pruži računar. On je u stanju da "zapamti", "obradi" i "ispiše" velike količine podataka znatno brže i kvalitetnije nego čovjek. U ovom [illegible]

će ukratko biti opisan razvoj informacionog sistema i korišćenje računara za rješavanje naprijed navedenih problema.

2. GRUPISANJE PROBLEMA I PODATAKA

Da bi se mjerna oprema mogla koristiti za provjeru kvaliteta proizvoda ili za utvrdjivanje nekih parametara, ona prethodno mora biti verifikovana kao ispravna. Te verifikacije se moraju vršiti u tačno odredjenim vremenskim intervalima (period verifikacije). Što je više mjerne opreme, to je teže pratiti kada koji instrument treba da se baždari. Tako dolazimo do prvog problema koji se postavlja pred rukovodioce a to je izrada plana baždarenja ili pregleda isteka verifikacija (slika 1).



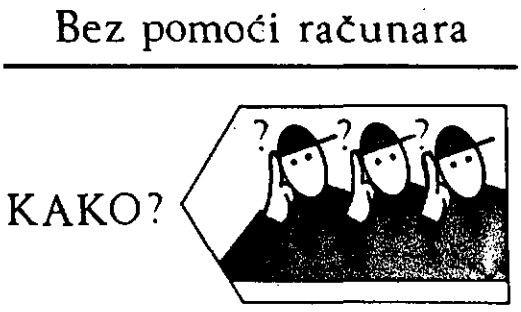
Slika 1

Da bi se moglo pristupiti računarskom rješavanju ovog problema mora se definisati jedan niz podataka: naziv, vlasnik, period verifikacije, i td. Ovu grupu podataka nazivamo STATIČKI PODACI O INSTRUMENTU jer su to podaci koje je dovoljno jednom prijaviti za svaki instrument i oni se nakon toga ne mijenjaju.

Kada neposredni izvršilac dobije instrument na baždarenje on mora da razmisli o tome kakav će postupak baždarenja izabrati, koju će mjernu opremu koristiti, koje će sve podatke o baždarenju upisivati, itd. Znači, prvi problem koji se postavlja pred neposrednog izvršioca je kako vršiti baždarenje (slika 2).

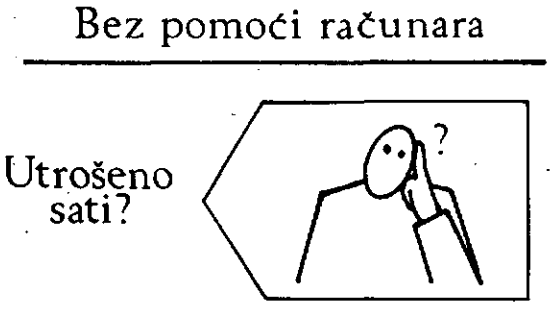
Za rješavanje ovog problema takodje se mora definisati niz podataka: uputstvo, karakteristika koja se ispituje, klasa tačnosti, mjerne tačke, itd. Ovu grupu podataka nazivamo STATIČKI PODACI O POSTUPKU BAŽDARENJA jer su to podaci o baždarenju instrumenata koji su trajni, tj. ne mijenjaju se i važe za sva baždarenja instrumenta.

Ove dvije grupe podataka nazivamo zajedničkim imenom STATIČKI PODACI.



Slika 2

Po završetku odredjene grupe baždarenja, zadatak rukovodilaca je pravljenje različitih analiza i pregleda izvršenih baždarenja. Na primjer, potrebno je odrediti koliko je vremena utrošeno na baždarenja instrumenata odredjenog vlasnika u odredjenom periodu (slika 3).



Slika 3

Kao i u prethodnim slučajevima, mora se definisati niz podataka koji su potrebni za rješavanje ovog problema: ko je baždario, kada, koliko je vremena utrošio, koju je opremu koristio, itd. Ovo su OPŠTI DINAMIČKI PODACI.

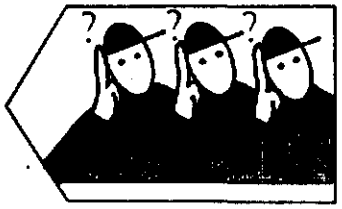
Posao neposrednog izvršioca se ne završava upisivanjem rezultata baždarenja. On još mora da izvrši odgovarajuće proračune i da donese odluku o tome da li instrument zadovoljava klasu tačnosti ili ne (slika 4).

Dodatni podaci koji su potrebni da bi se ovaj proračun mogao izvršiti automatski su DINAMIČKI PODACI O REZULTATIMA BAŽDARENJA, tj. izmjerene vrijednosti.

Ove dvije grupe podataka čine DINAMIČKE PODATKE koji se prijavljuju za svako baždarenje posebno.

Bez pomoći računara

Instrument zadovoljava klasu tačnosti?



Slika 4

3. STANDARDIZACIJA ZAPISNIKA

Osnovni dokument koji se koristi u metrološkoj laboratoriji je zapisnik o baždarenju. Značajan problem prilikom razvoja metrološkog informacionog sistema je predstavljala činjenica da zapisnici nisu bili standardizovani. Skoro da nije potrebno ni objašnjavati da je standardizacija zapisnika bila prvi neodložan korak u razvoju sistema. Ovo je posebno došlo do izražaja pri automatskom generisanju zapisnika, jer se u protivnom računar sigurno ne bi mogao iskoristiti na optimalan način. Ova standardizacija je vršena u dva dijela:

- standardizacija postupaka baždarenja, i
- standardizacija proračuna rezultata baždarenja.

Prvo su pažljivo proučene sve metode koje su se koristile, utvrdjeno je koje su im karakteristike zajedničke, a gdje su se razlikovale. Nakon toga je definisano desetak osnovnih tipova postupaka baždarenja i 4-5 osnovnih tipova proračuna rezultata baždarenja. Pri tome je ostavljena mogućnost da se odredjene grupe podataka mogu ponavljati više puta, zavisno od samog instrumenta. Na primjer, u istoj mjernoj tački može se vršiti jedno ili više mjerenja.

Sâm zapisnik se sastoji od dvije osnovne cjeline:

- naslovna strana, koja je jednaka za sve tipove zapisnika i u svakom zapisniku se pojavljuje samo jednom, i
- ostale strane sa rezultatima baždarenja, koje su različite zavisno od tipa zapisnika i kojih može biti proizvoljan broj i proizvoljna kombinacija za svaki pojedinačni zapisnik.

Postupak baždarenja, tj. izgled zapisnika o baždarenju, se mora definisati unaprijed za svaki instrument ako se želi pomoć računara. Za svako sljedeće baždarenje taj postupak ostaje isti, osim u izuzetnim slučajevima (promjena tehnologije baždarenja) kada se postupak izmjene mora sprovesti na poseban način.

Postoje dvije osnovne vrste zapisnika:

- radni, i
- izvještajni.

Radni zapisnik je dokument koji dobija izvršilac baždarenja prije samog baždarenja. Statički podaci o instrumentu i statički podaci o postupku baždarenja su već upisani ("upisao" ih je računar), te ovaj dio posla ne mora da uradi neposredni izvršilac. Šta više, on kroz ove podatke dobija kompletne upute kako da vrši baždarenje i već ima rezervisan i tačno definisan prostor za upisivanje rezultata samog baždarenja. Naravno, rezultati baždarenja se na ovaj dokument upisuju ručno. U sljedećem koraku se ti rezultati predaju računaru i on vrši odredjeni proračun. Koji će se tip proračuna primijeniti, zavisi od definisanih statičkih podataka. Nakon toga, računar može da generiše i izvještajni zapisnik. Svi podaci u izvještajnom zapisniku su "upisani" iz računara. Osim podataka koje je upisivao za radni zapisnik, sada postoje još i rezultati mjerenja, kao i rezultati proračuna sa tačno označenim vrijednostima koje prelaze dozvoljene granice (ako ih ima). Računar takode generiše i zaključak o tome da li instrument zadovoljava klasu tačnosti ili ne, tj. da li se može verifikovati kao ispravan.

4. IDENTIFIKACIJA ZAPISNIKA

Svaki instrument koji dolazi na baždarenje dobija jedinstvenu identifikaciju (ident-broj). Nakon toga, za taj instrument se mogu definisati i ostali podaci (naziv, tip, postupak baždarenja, itd.). Da bi se mogli preuzeti rezultati baždarenja za taj instrument sa radnog zapisnika, neophodno je da i svaki zapisnik ima svoju jedinstvenu identifikaciju. Veza zapisnika sa odgovarajućim instrumentom se ostvaruje preko ident-broja (kad se jednom definiše, on se više ne može promijeniti. U tekućoj godini može se vršiti jedno ili više baždarenja instrumenta (zavisno od perioda verifikacije). Ovo nam omogućuje da na jednostavan način dobijemo jedinstven identifikacioni broj svakog zapisnika - uz ident-broj vežemo dvije zadnje cifre godine u kojoj se vrši baždarenje i redni broj baždarenja u toj godini. To ujedno omogućuje i lakše traženje arhivskog zapisnika (zna se za koji instrument, u kojoj godini i po kojem rednom broju je vršeno baždarenje čiji se zapisnik traži). Da bi se automatski mogao generisati broj sljedećeg radnog zapisnika, osim ident-broja instrumenta koji se baždari neophodno je znati godinu i broj koji definišu broj zadnjeg prijavljenog izvještajnog zapisnika. Kako su ovo sve podaci koji su poznati računaru u trenutku generisanja radnog zapisnika, to on generiše i broj zapisnika. Ako se rezultati baždarenja prijavljuju računaru oni se moraju vezati uz taj broj i taj broj postaje i broj izvještajnog zapisnika.

5. PROJEKTOVANJE BAZE PODATAKA

Kao polazna metoda u projektovanju struktura podataka korištena je ISAC-metoda. Izmedju dva moguća rješenja (indeks-sekvencijalne datoteke i baza podataka) izabrano je modernije - baza podataka. Baza podataka je, u stvari, dinamički model nekog realnog sistema i mora da obezbijedi konciznu, jasnu i vjernu predstavu stanja sistema u bilo kom trenutku vremena. Da bi se to moglo obezbijediti, neophodno je da postoji kontrolisan pristup dodavanju novih, te modifikovanju i izdavanju postojećih podataka u bazi. S druge strane, neophodno je da podaci budu memorisani na način neovisan od programa koji ih koriste. Kao što je izvršena standardizacija zapisnika u cilju lakšeg rada sa zapisnicima, tako je izabran i sistem za upravljanje bazom podataka kao jedan standardizovani aparat za rad sa podacima. Podaci su opisani na jedinstven način (šemom i podšemom), te tako nema potrebe da se za svaki novi program ponovo opisuju. Za pristup tim podacima programi koriste odgovarajuće iskaze jezika za manipulisanje podacima. Svi podaci su smješteni i čuvaju se na jednom mjestu, što smanjuje mogućnosti redundantnosti i nekonzistentnosti podataka. Izborom baze podataka održavanje veza medju pojedinim podacima prepušteno je sistemu za upravljanje bazom podataka.

Već u prvim koracima sagledane su dvije osnovne grupe podataka koje su sada (u terminima baza podataka) smještene u dvije osnovne zone:

- zona statičkih podataka o instrumentu i postupku baždarenja, i
- zona dinamičkih podataka o baždarenju.

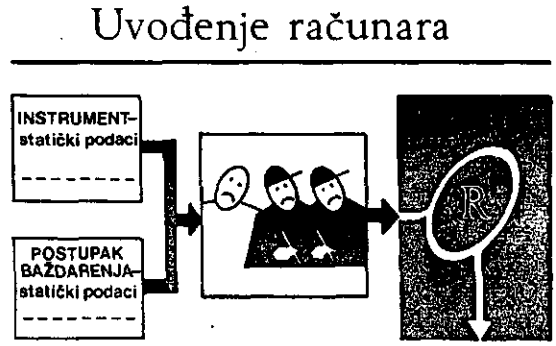
U svakoj grupi podataka definisane su manje logičke cjeline kao i veze medju njima. To je omogućilo da se definišu tipovi slogova i tipovi skupova metrološke baze podataka (šema).

Vidjeli smo da su problemi u radu u metrološkoj laboratoriji podijeljeni u dvije osnovne grupe - problemi rukovodilaca i problemi neposrednih izvršilaca. Tako je i dalji razvoj sistema podijeljen u dva osnovna pravca, razvoj podsistema za planiranje i praćenje izvršenih baždarenja i razvoj podsistema za rad sa zapisnicima. Za svaki od ovih podsistema može se definisati posebna podšema, pri čemu podšema podsistema za rad sa zapisnicima mora obuhvatiti kompletnu šemu (tj. ovaj podsistem koristi sve definisane podatke).

U konkretnoj realizaciji, fizički kapacitet baze podataka je takav da je ona smještena na jednom disku što omogućuje direktan pristup količinama od 30.000 instrumenata, odnosno 60.000 zapisnika. Kada se ovaj kapacitet popuni, neaktivni instrumenti (oni koji se trenutno ne baždare) i/ili najstariji zapisnici se mogu prebacivati na trake i čuvati po potrebi. Na ovaj način se oslobadja potreban kapacitet za prijavu novih instrumenata i/ili zapisnika. Realizovana baza podataka je mrežnog tipa (IDS/II). Za rad sa bazom podataka mogu se koristiti ili batch ili on-line programi (svi pisani u COBOL-u). On-line verzija je radjena u TDS-u i koristi se uglavnom za direktne upite i provjere. Za prijavu podataka i generisanje izvještaja koristi se uglavnom batch verzija. Podaci i zahtjevi za izvještaje se unose u računar preko posebnih maski za unos.

6. INICIJALNO PUNJENJE BAZE PODATAKA

Kada su definisani svi potrebni podaci oni su podijeljeni u manje grupe (transakcije). Za svaku transakciju je definisan poseban obrazac za unos. U inicijalnom upisu se odmah krenulo sa upisom statičkih podataka, jer se oni moraju prvi prijavljivati. Oni su prvo upisivani na odgovarajuće obrasce, zatim su se prenosili na trake i potom su ubacivani u bazu podataka pomoću programa za upis statičkih podataka. Ovo je kompleksan posao koji zahtijeva dosta kapaciteta (ljudskih, računarskih - slika 5). Organizovan je u sukcesivnim koracima, što znači da je jedna po jedna grupa instrumenata prolazila opisanu proceduru. Nakon što su ispravljene i sve eventualne greške koje su se desile prilikom unosa, za datu grupu instrumenata se moglo početi i sa prijavljivanjem dinamičkih podataka.



Slika 5

7. EFEKTI PRIMJENE

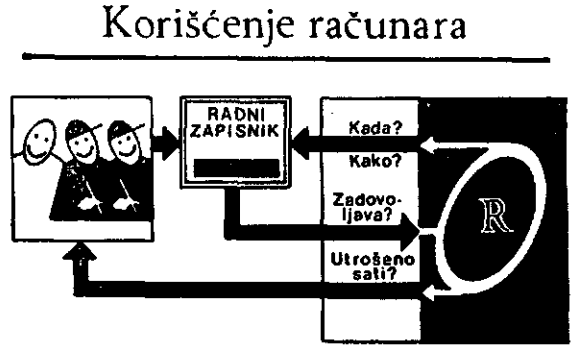
Nakon inicijalnog upisa podataka o instrumentu, računar prati datume verifikacija i upozorava kada će isteći periodi verifikacija za pojedine instrumente. Tako je omogućeno:

- kvalitetnije planiranje, koje doprinosi smanjenju zastoja u proizvodnji (na vrijeme se može predvidjeti kada će mjerna oprema biti na baždarenju),
- blagovremeno baždarenje instrumenata, koje obezbjedjuje sigurnost i tačnost mjerne opreme koja se

koristi, a samim tim i preciznije zadovoljenje zahtjeva u pogledu kvaliteta završnog proizvoda.

Nakon inicijalnog upisa podataka o postupcima baždarenja pojedinih instrumenata postignuto je:

- izvršilac baždarenja dobija gotove upute za baždarenje i sve ostale neophodne podatke u vidu radnog zapisnika,
- računar vrši proračun rezultata baždarenja i generiše izvještaje,
- povećan je i kvalitet i kvantitet obavljenih baždarenja, te oslobodjeno vrijeme za kreativan rad,
- znatno brže se dobija izvještaj o nekom obavljenom baždarenju ili pregled o obavljenim baždarenjima,
- oslobodjen je i prostor koji je bio potreban za čuvanje zapisnika.



Slika 6

8. ZAKLJUČAK

U razvijenom metrološkom informacionom sistemu rješavanje svih problema koji su opisani u poglavlju 2 (kada, kako, utrošeno, zadovoljava) je prepušteno računaru (slika 6). Računar izdaje preglede isteka verifikacije u kojima je predvidjen prostor za definisanje datuma dolaska na verifikaciju. Datum dolaska na verifikaciju se definiše zavisno od datuma isteka verifikacije, stvarnih potreba za korišćenje instrumenta koji se treba verifikovati i raspoloživih kapaciteta (neposredni izvršioci koji vrše verifikaciju, oprema koja se koristi pri verifikaciji). Nakon definisanja datuma dolaska na verifikaciju izdaju se planovi baždarenja i radni zapisnici po tim planovima. Rezultati izvršenih baždarenja se vraćaju računaru, te on nakon toga vrši odgovarajući proračun, izdaje izvještajne zapisnike i preglede obavljenih baždarenja.

Važno je još napomenuti da je razvijeni sistem u svakodnevnoj eksploataciji u jednoj proizvodnoj radnoj organizaciji. Postignuta je dnevna ažurnost, tj. sve promjene koje se dese u toku dana se istog popodneva unose u bazu i zatim korisnik dobija odgovore na pitanja koja je postavio.

U praksi su poznati slučajevi razvijanja računarski baziranih sistema koji prate samo osnovne podatke o instrumentima i eventualno omogućuju jednostavnije planiranje. Medjutim, u ovom trenutku ne postoji komercijalno poznat sistem koji bi pokrivao i automatizaciju proračuna rezultata baždarenja, te je tim značajniji opisani METROLOŠKI INFORMACIONI SISTEM.

9. LITERATURA

1. M.Lundeberg, G.Goldkuhl, A.Nilsson: INFORMATION SYSTEMS DEVELOPMENT - A SYSTEMATIC APPROACH, July 1978.
2. M.Arbutina: PROJEKTOVANJE RAČUNARSKI BAZIRANOG INFORMACIONOG SISTEMA ZA POTREBE RAZVOJA MJERNOG SISTEMA, ETF - Sarajevo, magistarski rad, septembar 1984.
3. E.Crnovršanin: PROJEKTOVANJE TRANSAKCIJSKOG SISTEMA ZA POTREBE RAZVOJA METROLOŠKOG INFORMACIONOG SISTEMA, ETF - Sarajevo, magistarski rad, septembar 1984.
4. --- : MJERNI SISTEM JUGOSLAVIJE, JUKEM Savez društava za mjernu tehniku Jugoslavije, jun 1983.
5. M.Arbutina: RAZVOJ BAZE PODATAKA METROLOŠKOG INFORMACIONOG SISTEMA, Primena računara u vazduhoplovnim zavodima - Savetovanje Beograd, decembar 1984.
6. E.Crnovršanin: TRANSAKCIONI SISTEM ZA POTREBE RAZVOJA METROLOŠKOG INFORMACIONOG SISTEMA, Primena računara u vazduhoplovnim zavodima - Savetovanje Beograd, decembar 1984.

PAKET ZA PROJEKTIRANJE PANORAMSKIH PLOŠČ

M. Toni, R. Kolar, M. Marušič, U. Marušič, M. Gril
Institut Jožef Stefan, Ljubljana

UDK: 681.3.06

Članek opisuje paket za daljinsko kontrolo. Namen paketa je avtomatizacija ročnih postopkov v procesu načrtovanja panoramske plošče pri signalno varnostnih napravah na železniški postaji. Zasnova paketa je neodvisna od področja, kjer je uporabljen, zato so opisana osnovna pravila, ki so nas vodila pri realizaciji paketa.

The paper describes the Remote Control Package.The purpose of the package is the automation of the manual procedure in the process of designing display panel for railway station signaling safety equipment. The concept of the package is completely independent of the field where it is currently applied. The basic principles that govern the approach of our package are described.

UVOD

Aplikacija, ki jo opisujemo v tem članku je nastala v okviru sodelovanja med Iskro Avtomatiko, Tozd Sistemi in Institutom Jožef Stefan, Odsek za uporabno matematiko. Je le del večje naloge, o kateri smo poročali že na drugih mestih /1//2/. Pokriva eno izmed področij delovanja Iskre Avtomatike, Tozd Sistemi. To je organizacija, ki projektira in izvaja dela za različne sisteme, ki zagotavljajo varnost v železniškem in cestnem prometu. Zaradi obsežnosti del in širokem strokovnem znanju, ki ga mora imeti projektant se je že zelo zgodaj pokazala potreba, da se projektanta razbremeni rutinskih del, s katerimi se ukvarja pretežni del svojega delovnega časa. S tem se omogoči, da se več časa ukvarja z ustvarjalnim delom. Z avtomatizacijo preprostih, a dolgotrajnih postopkov bi lahko razbremenili ljudi z obsežnim znanjem, ki največkrat predstavljajo ozko grlo v celotnem procesu. Več časa bi lahko posvetili zahtevnejšim opravilom, kar bi pripomoglo k večji kvaliteti dela in hitrejšemu dokončanju projekta.

Dolgoročni cilj sodelovanja med Iskro Avtomatiko, Tozd Sistemi in Institutom Jožef Stefan, Odsek za uporabno matematiko, je avtomatizacija ročnih postopkov pri projektiranju standarnih projektov. Pod pojmom standardni projekt mislimo na projekte, kjer postavljamo in povezujemo standardne elemente na način, ki je odvisen od konkretne situacije. Na način postavljanja, oziroma na alogritem, vpliva oblikovanost terena,posebne zahteve, velikost objekta in podobno. Lastnosti standardnih elementov so dobro določene, saj so to tovarniški izdelki, ki se na terenu le še vgradijo in povežejo s celotnim sistemom. Lastnosti standardnega elementa se ne spreminjajo, pogosto pa se pojavijo novi elementi z drugačnimi lastnostmi.

Izhodi takšnega standardnega projekta so lahko zelo obsežni in kompleksni. Navadno so sestvaljeni iz različnih izračunov, tehničnih risb, navodil za montažo, dokumentacije za vzdrževanje, opisov opreme, seznamov opreme in stroškov ter drugih tekstov.

Glavni problemi so predvsem zaradi obsežnosti dokumentacije in kratkih rokov, v katerih jo je potrebno izdelati. Dober primer je povezava več desettisoč sponk. Napake se pokažejo šele na terenu, kjer je odpravljanje drago in dolgotrajno. Problem so tudi spremembe, ki nastopijo, ko je del dokumentacije že izdelan. Pri risbah, ki so dolge tudi več kot deset metrov, si lahko pomagajo z rezanjem in lepljenjem, vendar tega ni mogoče večkrat ponoviti. Pri tipkani dokumentaciji so problemi podobni. Že zaradi manjših sprememb v nalogi, se dokumentacija tako spremeni, da se največkrat ni mogoče izogniti ponovnemu tipkanju celotne dokumentacije ali le njenega dela. Kasnejše vključevanje ali opuščanje naprav v projektu največkrat pomeni, da se poruši celotni koncept povezovanja naprav v sistem. To povzroči težave pri montaži in vzdrževanju takšnega sistema.

Paket je bil zasnovan na večjem problemu, kot je aplikacija opisana v članku. Celotni paket lahko razdelimo na dva dela:

a.) splošni del, ki je uporabljiv v vseh aplikacijah in je sestavljen iz osnovnih orodij, kot so:

- tekst procesor
- grafični moduli
- sistem za kabliranje
- sistem za podatkovno bazo

b.) aplikacijski del, v katerega so zakodirana pravila in lastnosti posamezne aplikacije. Stopnja avtomatizacije je odvisna od kompleksnosti tega dela.

Takšna delitev računalniškega paketa omogoča postopnost avtomatizacije paketa in njegovo široko uporabo. Na ta način smo dosegli, da lahko v relativno kratkem času izvedemo avtomatizacijo procesa s podobno strukturo izhodnih dokumentov. Za novo aplikacijo je potrebno narediti le module, ki pripravijo podatke za splošni del paketa, potem pa jih obdelujemo s splošnimi moduli. Module, ki so specifični za aplikacijo, lahko izdelujemo in uvajamo postopoma. Tako proces avtomatizacije lahko razbijemo na več delov, ki jih rešujemo po korakih, med katerimi lahko ocenjujemo doseženi uspeh. V začetni fazi realiziramo izdelavo rezultatov ob šibki avtomatizaciji, ki pa jo postopoma povečujemo. To ustreza uporabniku, ki potrebjuje rezultate v končni obliki že v razmeroma kratkem času. Prednost reševanja problema od zadaj je tudi v tem, da lahko uporabnik

aktivno sodeluje pri novem projektu od samega začetka.

Da bo paket kar najbolj uporaben, smo definirali osnovna pravila, ki so:

- zajem podatkov mora biti kar se da enostaven in blizu uporabnikovemu načinu dela. To pomeni, da mora biti predstavitev podatkov podobna tisti pri ročnem načinu dela, saj le tako lahko skrajšamo čas, ki je potreben, da projektant samostojno dela s sistemom. Prav tako moramo onemogočiti večkratno zajemanje istih podatkov, da se izognemo redundancam. Za kasnejšo uporabo so vsi podatki na voljo iz podatkovne baze.

- Lastnosti vseh elementov, ki jih uporabljamo v fazi projektiranja, so zajeti v šifrantih, ki jih vzdržuje uporabnik. Elemente izberemo s šifro iz šifranta, zato se hkrati priredijo tudi vse nespremenljive lastnosti elementa. Tako smo uporabniku skrajšali čas dela,ki je potrebno za komunikacijo z računalnikom. Dodajanje novih elementov v šifrant je pri opisanemu sistemu relativno redko.

- Algoritmi za projektiranje se pogosto menjajo, zato mora biti spreminjanje algoritma enostavno. Algoritmi so parametrizirani v največji možni meri. Del parametrov, ki se pogosteje spreminjajo, vnese projektant interaktivno med samim izvajanjem programa. Ostali so zakodirani v podatkovni bazi in jih programi čitajo neposredno. Sistem je načrtovan tako, da projektantu ni potrebno vnašati parametrov, ki se poredko menjajo.

- V vseh fazah avtomatičnega generiranja rezultatov morajo biti možni ročni posegi. V večini projektov, ki jih avtomatsko obdelujemo, nastopi več izjem, ki jih je nemogoče predvideti ali celo posplošiti njihovo obravnavanje. Procesiranje in odločitve v zvezi s takšnimi izjemami smo prepustili projektantu, saj bi to bilo nemogoče reševati avtomatsko. Poleg tega se izjeme ne ponavljajo in je zato neprimerno takšne probleme prepustiti avtomatičnemu sistemu. V praksi to pomeni, da projektant preveri vmesne rezultate med posameznimi fazami in vključi oziroma spremeni podatke, ki so posledica izjem.

DALJINSKA KONTROLA

Pod sistemom daljinske kontrole razumemo centralno povezavo več železniških postaj v enoten sistem, kar naj omogoča večjo varnost in propustnost železniškega prometa.

Signalno varnostne naprave vsake posamezne železniške postaje so zaključen sistem, ki komunicira le s sosednjima postajama. Elementi signalno varnostnih naprav v železniški postaji so:

- zunanji elementi, kot so signali, kretnice, izolirani odseki, kabelske omarice in podobno

- postavljalna plošča ali postavljalna omarica, ki prikazuje vse zunanje naprave in njihovo stanje. S pomočjo tipk na tej napravi se ureja promet na železniški postaji. Postavljena je v poslopju železniške postaje, sestavljena pa je iz mozaikov

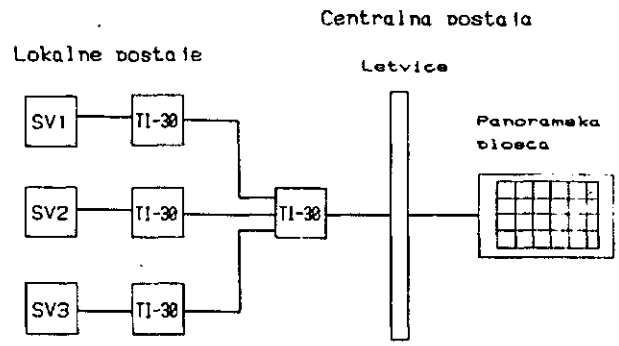
- kontrolna logika, ki preprečuje, da na postaji ne pride nesreče ali drugačne škode. To so relejne skupine v poslopju železniške postaje

- povezave med temi elementi

Vse naprave so med seboj povezane z kabli različnih presekov in tipov. Vsaka, nekoliko večja postaja ima nekaj deset tisoč povezav med sponkami. Povezave so dolge od nekaj centimetrov pa do nekaj kilometrov. Očitno je, da že majhne spremembe v lokaciji elementov ali v načinu povezovanja lahko povzročijo velike spremembe v stroških realizacije signalno varnostne zaščite na postaji.

Tipična operacija na železniški postaji je vzpostavitev vozne poti. To pomeni da hočemo nastaviti vse kretnice in signale tako, da bo mogoč dovoz na želeni tir. Seveda je takšen tir potrebno tudi zaščititi pred prihodom drugega vlaka, tudi če gre za pomoto. S pritiskom na ustrezne tipke postavljalni omarici sistem najprej preveri, da kakšen od tirov na odseku ni že zaseden, nastavi na "stop" vse signale, ki vodijo na ta odsek, omogoči varnostne poti, nastavi kretnice v ustrezno lego in postavi signale na "prosto" za dovoz na odsek.

Daljinska kontrola je nadgradnja signalno varnostnih naprav. Na centralnem mestu povezuje in koordinira več lokalnih postaj. Na ta način se poveča propustnost prometa, kakor tudi zanesljivost in varnost.



Centralna postaja daljinske kontrole sprejema podatke o stanju zunanjih naprav v lokalnih postajah. Podatke sprejema in obdeluje s pomočjo računalnika TI-30. Poleg tega panoramska plošča na steni prikazuje situacijo celotne proge.

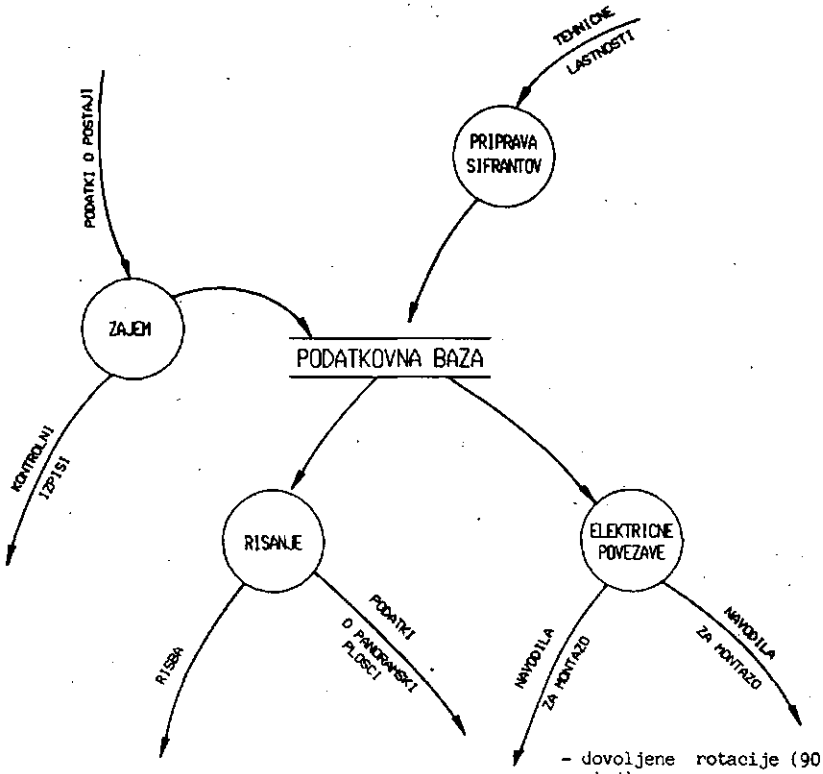
Panoramska plošča je prikazovalna naprava, ki je lahko velika tudi nekaj deset metrov. Sestavljena je iz več tisoč mozaikov velikosti 40X40 mm. Prikazuje stanje na celotni progi in vseh lokalnih postajah, ki jih nadzira. Vsak mozaik prikazuje delček postaje ali proge. Grafično ponazarja tir in napravo na tiru. Poleg tega vsebuje še oznako naprave in svetlobne indikatorje, ki prikazujejo v kakšnem stanju je prikazovana zunanja naprava. Ti svetlobni indikatorji so različnih barv.

Primer indikatorjev za zunanjo napravo:

Kretnica	+stanje, -stanje, faza premikanja, poškodovana (prerezana)
Signal	prosto, stoj
Izoliran odsek	prosto, zasedeno

OPIS PAKETA RC - DALJINSKA KONTROLA

Delo na celotni nalogi se je začelo leta 1978 kot sodelovanje med Iskro Avtomatiko, Tozd Sistemi in Fakulteto za elektrotehniko. Kasneje se je priključil Institu Jožef Stefan, Odsek za uporabno matematiko in je delo prevzel v celoti. V posameznih fazah je bilo pri izvajalcu na projektu hkrati zaposleno od 3 do 10 ljudi.



Projekt RC - daljinska kontrola, ki je le del te celotne naloge, se je začel v drugi polovici leta 1983 in je bil dokončan v letu 1984, ko je bil tudi predan v redno uporabo. Od takrat mu je bil dodan le še grafičen zajem panoramske plošče, ostalih sprememb pa ni bilo treba dodatno vgraditi.

Paket je sestavljen iz približno 50 programov, ki imajo skupaj okrog 75000 vrstic kode v programskem jeziku PASCAL.

Po vsebini paket lahko razdelimo na 3 dele in to:

1. zajem podatkov
2. izdelava dokumentacije za panoramsko ploščo
3. izdelava dokumentacije električnih povezav

1. ZAJEM PODATKOV

Zajemamo podatke za vsako lokalno postajo posebej. Najprej zajamemo splošne podatke, kot so številka in ime postaje, ime projekta, ime proge, velikost postaje, abeceda, ki smo jo izbrali za označevanje in še več drugih parametrov.

Takoj nato že lahko zajamamo podatke za panoramsko ploščo. To projektant vnese na osnovi ročno izdelane skice. Na zaslonu vidimo del panorame, ki jo želimo spremeniti. Vsak mozaik na zaslonu je predstavljen s kodo mozaika iz šifranta mozaikov. Z ukazi premikamo okno po panoramski plošči in zajemamo podatke o mozaikih. Za vsak mozaik vnesemo le kodo mozaika, po potrebi pa še lego (če je mozaik obrnjen za 90,180 ali 270 stopinj) in oznako zunanje naprave, ki jo mozaik predstavlja. To so vsi podatki, ki jih moramo vnesti za določen mozaik.

Seveda je dovoljen le vnos tistih mozaikov, katerih lastnosti so shranjene v podatovni bazi. Tu so shranjene vse lastnosti mozaikov, ki se ne spreminjajo. To so:

a.) Parametri za zajem mozaika, ki povedo, katere podatke je pri mozaiku potrebno zajeti.
   - dovoljene rotacije (90,180 ali 270 stopinj)
   - velikost mozaika (npr 2x2 velikosti osnovnega mozaika)
   - število in dolžina oznak zunanjih elementov, ki jih predstavlja mozaik. Te oznake služijo za razločevanje zunanjih naprav, ki so istega tipa. Lahko se pojavljajo tudi na panoramski plošči in nam služijo tudi za identifikacijo sponk pri električnih povezavah.

b.) Grafični opis mozaika je zaenkrat še ročno zakodiran, ker se mozaiki spreminjajo relativno poredko. Za vsak osnovni grafični element mozaika vstavimo v podatkovno bazo po en zapis. Vsak mozaik je običanjo zakodiran z 3-10 zapisi. Grafični opis mozaika vsebuje:
   - osnovni grafični element (npr. signal, kretnica, tir...) in njegove relativne koordinate
   - koordinate in vrsta teksta za oznake zunanjih naprav, ki jih predstavlja mozaik.

c.) Opis električnih lastnosti mozaika definira način povezave mozaika z ostalimi napravami:
   - Simbolična imena zunanjih naprav, ki jih predstavlja mozaik
   - Identificacijske kode sponk mozaika in pripadnost zunanji napravi.

d.) Kode osnovnih mozaikov, ki sestavljajo sestavljeni mozaik.

e.) Dodatni podatki o mozaiku
   - Tekstovni opis mozaika
   - Podatki za izdelovalca mozaikov
   - Cena mozaika

Ko je faza vnosa zaključena, so nam znani praktično vsi podatki za izdelavo projektne dokumentacije. V poznejših fazah vnašamo le še parametre za krmiljenje obdelave in algoritmov.

2. TEHNIČNE RISBE

Grafični del paketa izdela več različnih tipov risb. Izbiramo lahko med risanjem celotne panorame, posamezne postaje ali risanjem z oznakami za izdelovalca mozaikov. Običajno risbo pomanjšujemo v različnih razmerjih, da je v standardnem formatu, največkrat v podaljšanem A4.

Tudi grafični del opreme lahko razdelimo na dva dela in to na del, ki je specifičen za daljinsko kontrolo in del, ki je splošen in ga uporabljamo tudi pri drugih aplikacijah.

a.) Specifični del paketa zgradi celotno sliko s vsemi elementi. Izdelajo se slike mozaikov, kakor tudi dodatnih elementov (glava risbe, mreža in oznake).

b.) Splošni del pa je sestavljen iz od naprav neodvisnih modulov in od naprav odvisnih modulov. Trenutno imamo drugi del realiziran za risalnike Versatec in Tectronix.

3. DOKUMENTACIJA ELEKTRIČNIH POVEZAV

Panoramsko ploščo sestavljata dve vrsti mozaikov: navadni mozaiki za prikazovanje stanja zunanjih naprav in mozaiki za prikazovanje številke vlaka. Vse mozaike krmili računalnik TI-30 in sicer: prve preko DIZK, kablov, N-delilnika in konektorjev, druge pa direktno preko KRDISE.

V tej fazi izdelamo dokumentacijo za montažo elementov, kakor tudi za vzdrževanje. Dokumentacija za montažo je izdelana za vsak konec kabla na svojem listu papirja, da ima montažer lažje delo. Dokumentacija za vzdrževanje omogoča sledenje posameznega signala od mozaika do TI-30. Dokumentacijo za navadne mozaike in dokumentacijo za mozaike, ki prikazujejo številko vlaka, izdelamo posebej.

ZAKLJUČEK

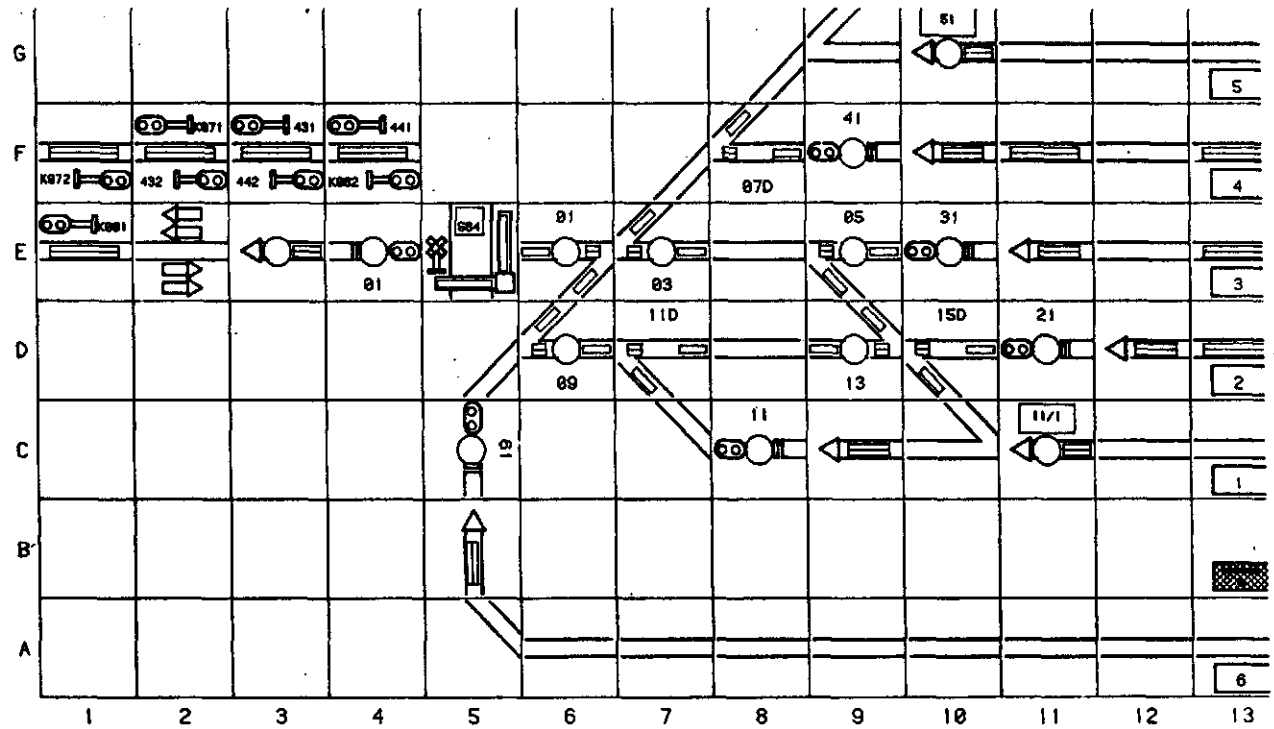
Paket je v uprabi približno leto in pol. V tem času je bila izdelana celotna dokumentacija za daljinsko kontrolo za več kot 70 železniških postaj. Za delo s paketom je pri uporabniku usposobljenih več sodelavcev, ki delajo popolnoma samostojno. V času, ko je paket v uporabi, smo ugotovili naslednje prednosti, ki jih ima delo s paketom v primerjavi z ročnimi postopki:

- Približno 6 krat krajši čas izdelave projektne dokumentacije
- Sodelavce za delo s paketom je neprimerno lažje izšolati kot za delo z ročnimi postopki
- Večja kvaliteta izdelka
- Manj napak
- Neprimerno hitrejša in kvalitetnejša realizacija sprememb, ki se pojavijo pozno v projektu. Pri ročnem postopku je bilo večkrat potrebno ponoviti celoten postopek.
- Večje število projektov, ki so v obdelavi istočasno

REFERENCE:

/1/ B. Vilfan, R. Kolar, M. Toni, M. Marušič, V. Mahnič, CAD in the Design of Railway-Highway Intersection Safety Equipment, EUROGRAPHICS 83, Zagreb

/2/ M. Toni, R. Kolar, M. Marušič, U. Marušič, M. Gril, Panel Design for Railway Station Signaling Safety Equipment, Austro-Yugoslav Conference on Computer Graphics, 1984



MIKRORAČUNALNIŠKI SISTEM ZA PRENOS IN OBDELAVO INFORMACIJ PRI POŽARNI ZAŠČITI - PASO REKA

J. Plavc, M. Maher, B. Črnivec, B. Delak, J. Zajc,
Iskra Avtomatika, TOZD Sistemi, Ljubljana

UDK: 681.3.014

Povzetek:

PASO je mikroprocesorski sistem za požarno zaščito objektov. Zaradi svojega modularnega koncepta nudi veliko fleksibilnost za posamezne aplikacije. Glavni trije nivoji sistema so: zbirne postaje, koncentratorji in nadzorni centri. Prvi del članka opisuje vse možnosti sistema PASO, drugi del pa konkretno aplikacijo, ki že obratuje na Reki.

Summary:

PASO is a microprocessor system for fire protection of objects. Because of it's modular conception it offers a great flexibility for different applications. The main three levels of the system are: remote stations, concentrators and supervision centers. The first part of this paper describes all possibilities of the PASO system, the second part describes application that is already working in Reka.

1.0 Splošno o sistemu PASO

Mikroračunalniški sistem za požarno zaščito objektov ima nalogo, da posreduje informacije od ščitenih objektov do centralnega mesta, kjer jih mora predelati tako, da jih poda operaterju v ustrezni obliki. Sistem sestavljata aparaturni in programski del, ki sta sestavljena modularno, tako da dosežemo čimvečjo fleksibilnost pri načrtovanju konkretnih aplikacij. To nam seveda omogoča tudi kasneje, ko nek sistem že deluje, enostavno dopolnjevanje in razširjanje velikosti in funkcij sistema.

Aparaturni del sestavljajo aparaturni moduli teleinformacijskega sistema TI-30 in periferne enote. Sistem TI-30 omogoča zajem digitalnih informacij, njihov prenos in obdelavo za prikaz in shranjevanje. Uporabljena je družina osembitnih mikroprocesorjev M 6800 - Motorola (procesor 6802 ali 6808, ter ostala VLSI vezja družine M 6800). Vrsta in količina modulov ter perifernih enot se določi glede na količino ščitenih objektov ter glede na zahteve investitorja.

Programski del sistema za požarno zaščito sestavlja programski paket Iskra MOSPIK, ki je grajen modularno in transparentno. Modularnost programske opreme omogoča konfiguriranje programskih paketov, ki realizirajo različne funkcije. Programske module povezuje monitorski program. Programska oprema sistema PASO je transparentna, ker je delovanje in povezovanje posameznih programskih modulov neodvisno od njihove porazdelitve v mikroračunalniški mreži. Programski moduli se delé na sistemske in aplikacijske. Sistemski programski moduli so napisani v zbirniku ASM 6800, aplikacijski programski moduli pa predvsem v Pascalu. Z aplikacijskimi moduli dosežemo prilagoditev sistema konkretni aplikaciji.

Funkcijsko se sistem za požarno zaščito deli na tri nivoje:

- zbirni nivo, kjer se na zbirne postaje priključujejo ščiteni objekti,
- nivo koncentracije, ki ga tvori eden ali več koncentratorjev,
- nivo nadzora, kjer je eden ali več nadzornih centrov, ki omogočajo komunikacijo operaterja s celotnim sistemom.

Vsaka zbirna postaja, koncentrator ali nadzorni center je neodvisna mikroračunalniška enota z najmanj enim procesorjem. V koncentratorju sta lahko dva procesorja - koncentrator A in koncentrator B; za najbolj zahtevne aplikacije je v nadzornem centru možna lokalna mikroračunalniška mreža, ki z distribucijo aktivnosti omogoča večjo hitrost in zanesljivost delovanja.

Med zbirnimi postajami in koncentratorji ter med koncentratorji in nadzornimi centri so modemske zveze.

1.1 Zbirni nivo

Tu je lahko do osem zbirnih postaj priključenih na en koncentrator. Vsaka zbirna postaja je lahko priključena le na en koncentrator in predstavlja neodvisen mikroračunalnik, na katerega je priključeno do 256 ščitenih objektov.

Ščiteni objekt je s stališča mikroračunalniškega sistema lokalna požarna centrala. Le-ta pošilja v zbirno postajo dvobitno informacijo o svojem stanju:

- normalno stanje;
- alarm;
- motnja na zvezi;
- prekinitev linije.

Te informacije od ščitenega objekta do zbirne postaje se pošiljajo frekvenčno modulirane po standardnih telefonskih linijah, ne da bi bila motena običajna komunikacija po telefonski liniji. Zbirne postaje so navadno nameščene v postnih telefonskih centralah.

Naloge mikroračunalnika v zbirni postaji so naslednje:

- ciklično pregleduje stanja objektov;
- ob ugotovljeni spremembi stanja sestavlja sporočilo za koncentrator;
- testira delovanje lastne aparaturne opreme;
- na zahtevo koncentratorja izvede splošni poziv in pošlje stanja vseh objektov.

Cikel pregledovanja objektov je manjši od dveh sekund. Čas komunikacije med zbirno postajo in koncentratorjem ob spremembi stanja ščitenega objekta je odvisen od števila ugotovljenih sprememb v prvem ciklu. Tipično je ta čas 1 s, maksimalno (*ob spremembi stanja vseh objektov*) pa deset sekund.

1.2 Nivo koncentracije

Na tem nivoju imamo lahko največ štiri koncentratorje. Osnovna naloga koncentratorja je, da ranžira informacije iz posameznih zbirnih postaj do različnih nadzornih centrov. Tako je lahko en koncentrator priključen na več nadzornih centrov, največ na osem, kar je največje število nadzornih centrov.

Naloge mikroračunalnika na nivoju koncentracije so:

- sprejemanje spontanih sporočil iz zbirnih postaj;
- ranziranje informacij na različne nadzorne centre;
- pošiljanje sporočil nadzornim centrom;
- testiranje linij do zbirnih postaj;
- testiranje delovanja lastne aparaturne opreme na zahtevo nadzornega centra;
- pošiljanje stanja vseh objektov (splošni poziv).

Iz opisa nalog koncentratorja vidimo, da je zanesljivost delovanja koncentratorja zelo pomembna karakteristika. Zato je lahko v vsakem koncentratorju dvojnoračunalniška konfiguracija. To pomeni, da so tako aparaturni moduli kot programska oprema podvojeni. Prvi mikroračunalnik spremlja in vodi proces medtem ko drugi spremlja le delovanje procesorja prvega mikroračunalnika. V slučaju okvare vodečega mikroračunalnika se izvede preklop in proces začne upravljati dotedanji spremljajoči mikroračunalnik.

Med zbirnimi postajami in koncentratorjem je modemska komunikacija, tako da je potrebno za priključitev posamezne zbirne postaje ena telefonska linija.

Koncentrator je običajno nameščen v postni telefonski centrali, vendar je to predvsem iz praktičnih razlogov ne pa tennoloških.

1.3 Nadzorni nivo

Tu je en ali več nadzornih centrov, ki omogočajo komunikacijo operaterja s sistemom za požarno zaščito. Zgradba posameznega nadzornega centra je odvisna od zahtev investitorja. Možno je do osem nadzornih centrov.

V maksimalni konfiguraciji opravlja nadzorni center dve vrsti nalog:

- naloge v realnem času;
- pomožne naloge.

Naloge v realnem času prestavljajo osnovne funkcije nadzornega centra:

- sprejemanje sporočil z nivoja koncentracije;
- obdelava informacij in sestavljanje: obratovalnih protokolov, akcijskih protokolov in študijskih protokolov;
- prikaz informacij na perifernin enotan;
- sprejemanje ukazov, ki jih vnaša operater;
- izvrševanje ukazov;
- obdelava informacij za arhiviranje in arhiviranje le-teh;
- ciklično testiranje modemskih linij do koncentratorjev;
- ciklično testiranje lastne aparaturne opreme.

Pomožne naloge vključujejo naslednje funkcije:

- vnašanje informacij o novih objektih;
- sestavljanje ali popravljanje akcijskih protokolov;
- sestavljanje ali popravljanje študijskih protokolov;
- obdelava arhiviranih informacij za analizo;
- prikaz arhiviranih informacij.

Da je mozno realizirati vse zgornje naloge, je nadzorni center organiziran z več mikroračunalniki, ki so povezani v mrezo, tako da so funkcije distribuirane. Tako dosežemo hitrejšo obdelavo informacij in večjo zanesljivost sistema.

Prikaz oz. vnos informacij je mozen na naslednjih perifernih enotan:

- alfanumerični zaslon; na njem se prikazujejo obratovalni protokoli o vsen relevantnih dogodkin, ki jih sistem detektira;
- pisalnik; na njem se izpisujejo obratovalni protokoli ter akcijski protokoli;
- grafični barvni zaslon; na njem se prikazujejo slike objektov z bliznjo okolico, kar olajšuje intervencijo v primeru alarma na objektu;
- sinoptična plošča; na njej se prikazujejo stanja objektov,
- zunanje spominske enote; na njin se arhivirajo informacije;
- alfanumerična tastatura; omogoča operaterju vnos ukazov in informacij, ki so potrebne za delovanje sistema,
- funkcijska tastatura; omogoča operaterju vnos ukazov in informacij.

V najbolj okrnjeni izvedbi nadzornega centra potrebujemo za prikaz in vnos informacij le alfanumerični terminal in pisalnik.

Na pisalnikih, zaslonih terminalov ali grafičnih barvnih zaslonih se lahko izpisujejo oz. prikazujejo tri vrste protokolov:

- Obratovalni protokoli; ti vsebujejo čas, kraj in vrsto relevantnega dogodka; le-ta je lahko sprememba na objektu, pojav napake (spremembe) na sistemu PASO ali pa ukaz operaterja.
- Akcijski protokoli; ti so sestavljeni iz tekstnega in slikovnega dela in so namenjeni operaterju v nadzornem centru in gasilski ekipi, ki gre intervenirat; v tekstovnem delu je opis objekta z navodili za intervencijo, v slikovnem delu pa je shematično narisan ščiteni objekt z bliznjo okolico.
- Študijski protokoli; ti so identično enaki akcijskim, le da se prikazujejo na zahtevo; namenjeni so operaterjem in gasilskim ekipam za vadbo.

Nadzorni center je običajno nameščen v komandnem centru gasilske brigade ali pa na upravi za notranje zadeve.

2.0 PASO Reka

Mikroračunalniški sistem za požarno zaščito je bil glede na gornji opis v nekoliko okrnjeni obliki izdelan za gasilsko brigado na Reki. Sistem na Reki ima naslednjo konfiguracijo:

- dve zbirni postaji, na kateri je priključenih devetdeset ščitenih objektov;
- en koncentrator z dvojnoračunalniško konfiguracijo;
- en nadzorni center, ki je nameščen v gasilski brigadi.

V naslednjih fazah gradnje sistema na Reki je predvideno še pet zbirnih postaj in dva nadzorna centra.

Zbirni postaji sta nameščeni v avtomatskih telefonskih centralah:

- ATC Barčićeva - priključeno je 62 objektov;
- ATC Sušak - priključeno je 28 objektov.

Koncentrator je nameščen v ATC Kozala. Sestavljata ga dva podsistema z identično aparaturno in programsko opremo: koncentrator A in koncentrator B.

Nadzorni center je nameščen v komandnem centru gasilske brigade. Izvaja naslednje naloge:

- sprejemanje sporočil z nivoja koncentracije;
- obdelava informacij in sestavljanje: obratovalnih protokolov, akcijskih protokolov in študijskih protokolov;
- prikaz informacij na perifernih enotah;
- sprejemanje ukazov, ki jih vnaša operater;
- izvrševanje ukazov;
- ciklično testiranje modemskih linij do koncentratorjev;
- ciklično testiranje lastne aparaturne opreme.

Informacije se prikazujejo oz. vnašajo na naslednjih perifernih enotah:

- zaslon A; na njem se lahko izpisujejo obratovalni protokoli, akcijski protokoli in študijski protokoli;
- tastatura A; vnašajo se vsi sistemski ukazi;
- zaslon B, na njem se lahko izpisuje obratovalni protokoli in študijski protokoli;
- tastatura B; vnaša se le omejeno število sistemskih ukazov;
- pisalnika I in II; na njih se izpisujejo vsi obratovalni protokoli;
- pisalnik III; na njem se izpisujejo akcijski protokoli, ki se takoj predajajo vodji ekipe, ki intervenira;
- sinoptična plošča; na načrtu Reke se z lučkami prikazuje stanje ščitenih objektov.

Operater vodi akcije preko terminala (zaslona in tastature) A. V primeru izpada terminala A je predvidena nadomestna strategija - vlogo terminala A dobi terminal B. Zahteve po večji zanesljivosti so tudi vzrok za izpisovanje obratovalnega protokola na dva pisalnika.

3.0 Zaključek

Opisani mikroračunalniški sistem za požarno zaščito objektov je del širšega sistema integralne zaščite objektov, ki ga razvijamo in apliciramo v ISKRI Avtomatiki. Predstavlja primer specializiranega informacijskega sistema z vsemi njegovimi karakterističnimi obdelavami informacij: zajem, prenos na večje razdalje, distribuirana obdelava in prikaz na perifernih enotah. Posebej je potrebno poudariti, da je aplikacija že realizirana in da je sistem že predan investitorju, tako da je premoščen razkorak med snovanjem in razvojem na eni strani, ter konkretno postavitvijo sistema na drugi.

Literatura:

/1/ Mospik: Opis in navodila za apliciranje multiprocesorskega multiprogramskega paketa; ISKRA Avtomatika, TOZD Sistemi; interna dokumentacija st. M 99999225411.

/2/ Teleinformacijski sistem TI-30 za operativno vodenje procesov; zgradba in opis delovanja; ISKRA Avtomatika; interna dokumentacija st. 437 009 056.

/3/ Funkcijska specifikacija požarnega alarmnega sistema za Reko - PASO Reka; ISKRA Avtomatika, TOZD Sistemi, interna dokumentacija st. Z 00101221.521.

/4/ Uporabniški priročnik za PASO Reka; Iskra Avtomatika, TOZD Sistemi; interna dokumentacija st. Z 00101345.563.

MULTIMIKRORAČUNALNIŠKI SISTEM ZA VODENJE IN NADZOR HE MAVČIČE

Miha SKUMAVC, Jure PLAVC,
Iskra Avtomatika, Ljubljana, Jugoslavija

UDK: 681.3.014

POVZETEK

Sistem za nadzor in vodenje hidroelektrarne Mavčiče je razdeljen na pet podsistemov, od katerih vsak opravlja specifično in zaokroženo področje nalog. Z računalniškega stališča gre za mikroračunalniško mrežo, na katero je vezano pet samostojnih mikroračunalnikov. Poleg teh je na verigo posredno priključen tudi mikroračunalniški podsistem za zajem, obdelavo in pošiljanje vodomernih podatkov z rek Kokre in Bistrice v HE Mavčiče. V končni fazi bo tako koncipiran sistem vodenja in nadzora omogočal avtomatsko delovanje HE Mavčiče iz območnega centra, oz. iz hidroelektrarne preko podsistema LOKIN ali direktno ročno na agregatu. Izbor mesta vodenja je realiziran na podsistemu LOKIN.

MULTICOMPUTER SYSTEM FOR THE CONTROL AND SUPERVISION OF HYDROELECTRIC POWER STATION MAVČIČE

ABSTRACT

The system for the control and supervision of Hydroelectric Power Station Mavčiče is divided into five subsystems, each handling a specific and self-contained field of jobs. The computer part consists of a microcomputer network which includes five idependent microcomputers. Along with these, a microcomputer subsystem is indirectly connected to the chain, providing the acquisition, processing and transmission of hydrometric data from the rivers Kokra and Bistrica to the Mavčiče HPS. In its final phase, this control and supervision system will allow automatic operation of HPS Mavčiče from the area center or from power station, respectively, over the subsystem Lokin or manually, on the power generator. The choice of control position is implemented in the subsystem Lokin.

Koncept multimikroračunalniškega sistema nadzora in vodenja HE Mavčiče je shematsko prikazan na sliki 1. Iz slike je razvidna konfiguracija mikroračunalnikov na mreži, ki zagotavlja vodenje hidroelektrarne in komuniciranje z nadrejenim centrom. Obdelave, ki jih izvajajo posamezni podsistemi, so naslednje:

- KRON1 in KRON2; sta kronološka podsistema za zajem enobitnih informacij in opremljanje s točnim časom nastanka (ločljivost znaša 10 ms). Zajem informacij je razdeljen na dva podsistema zato, da dosežemo večjo hitrost zajema. Vsak od podsistemov zajema 150 informacij s točnim časom in 100 signalizacij s sekundnim časom. Obdelave, ki jih izvajata podsistema, so:
  - primerjava starega in novega stanja na vhodu in ugotavljanje spremembe,
  - spravilo informacij s časom v izbrano podatkovno strukturo (banka stanj podsistema),
  - sestava sporočil za oddajo v mikroračunalniško mrežo,
  - izdaja sporočil v izhodno podatkovno strukturo.

  Programska oprema je za oba podsistema enaka in je sestavljena iz programskih modulov MOSPIK (Mikroračunalniški operacijski sistem protokoliranja in krmiljenja) in izbranih aplikativnih programskih modulov.

- LOKIN; podsistem je namenjen za dialog med operaterjem in sistemom. Poleg tega zajema vse meritve v BCD kodi. Informacije, ki jih zajema posredno, iz mikroračunalniške mreže ali neposredno pripravi za prikaz na različnih perifernih enotah in generira ustrezne protokole na pisalniku. Poleg tega omogoča poseganje operaterja v delovanje celotnega sistema HE, preko sistema za vodenje. Opravlja torej funkcijo intiligenčne funkcijske tastature. Obdelave, ki jih izvaja podsistem LOKIN:
  - zajem informacij iz MR mreže,
  - zajem informacij iz procesa - BCD meritve,
  - osveževanje banke stanj celotnega sistema,
  - ugotavljanje sprememb stanj in statistične obdelave meritev (prekoračitve praga),
  - sestavljanje obratovalnih protokolov, merilnih protokolov, spiskov enobitnih in dvobitnih javljanj, spiskov prisotnih alarmov, vezan izpis in zapis po izpadu (post mortom rewiew),
  - izpisi na pisalnikih (2) in prikazi na VT100 terminalu,
  - sprejem in obdelava ukazov iz tastature terminala.

  Programsko opremo sestavljajo aplikacijski in sistemski moduli programskega paketa MOSPIK.

- KP/FW; podsistem je namenjen neposrednemu zajemu analognih meritev in dvobitnih informacij. Te informacije obdela in pošlje v mikroračunalniško mrežo. Hkrati skrbi za komunikacijo z nadrejenim nadzornim centrom v OCV Beričevo. Osnovne naloge KP/FW so naslednje:
  - zajem analognih meritev neposredno iz HE Mavčiče,
  - zajem dvobitnih položajnih vrednosti neposredno iz HE Mavčiče,
  - zajem stanj iz MR mreže:
    - signalizacije iz KRON1 in KRON2,
    - 16-bitne meritve BCD iz LOKIN,
  - osveževanje podatkovnih struktur,
  - izdaja informacij neposredno v HE Mavčiče,
  - priprava informacij in komunikacija z OCV Beričevo.

  Programska oprema podsistema KP/FW je sestavljena iz dveh skupin programskih modulov, ki

realizirata dvojno vlogo podsistema KP/FW:

- programski moduli, ki realizirajo končno postajo,
- programski moduli, ki realizirajo mikroračunalniško mrežo.

- KP/MBK; ločen podsistem za prenos meritev s vodomernih postaj Kokre in Bistrice. Ta podsistem je vezan na multimikroračunalniško mrežo preko koncentratorja v HE Mavčiče. Koncentrator meritve BCD preko relejne omare posredno izdaja na vhod podsistemov LOKIN in REG; s tem dosegamo večjo zanesljivost celotnega MR sistema.

  Programska oprema je sestavljena iz programskih modulov MOSPIK.

- REG; podsistem direktno vodi delovanje hidroelektrarne Mavčiče. Za izvajanje izbranih nalogov zajema informacije iz:

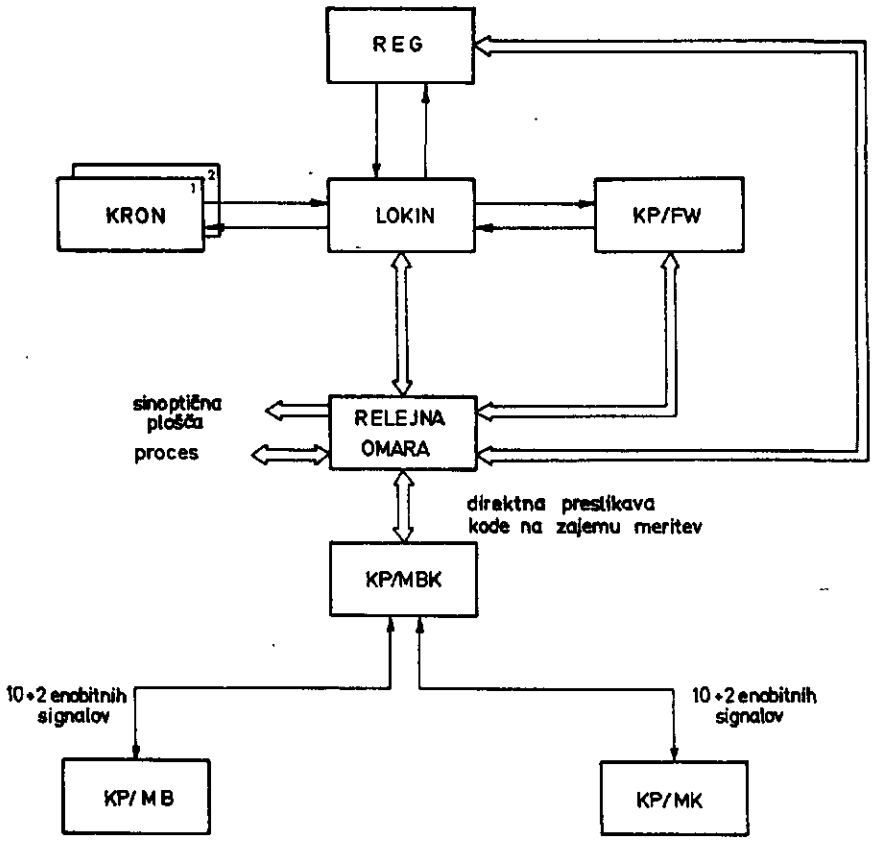
  - zajem digitalnih infbrmacij iz procesa,
  - zajem analognih meritev iz procesa,
  - zajem digitalnih meritev iz procesa,
  - zajem enobitnih javljanj (alarmi),
  - zajem informacij iz MR mreže - podsistema LOKIN,
  - zajem informacij iz MR mreže - podsistema KP/FW.

  Nad zajetimi informacijami REG izvaja sledeče obdelave:

  - obdelava naloge za obratovanje po moči:
    - vožnja z delovno močjo,
    - vožnja z jalovo močjo,
    - kontrola napetosti na zbiralkah in popravljanje jalove (v opciji),
  - obdelava naloge za obratovanje z nastavljenim pretokom,
  - obdelava naloge z nastavljenim nivojem,
  - obdelava naloge ugotavljanja razpoložljivosti lokalnih avtomatov na HE,
  - obdelava alarmnih stanj.

  Programska oprema podsistema REG je sestavljena iz programskih modulov MOSPIK in aplikacijskih modulov za vodenje HE Mavčiče.

Aparaturno opremo multimikroračunalniškega sistema za nadzor in vodenje HE Mavčiče sestavljajo moduli mikroračunalniškega sistema TI-30 Iskra Avtomatika.



Slika 1. Shema povezav multimikroračunalniškega sistema za vodenje HE

MULTIRAČUNALNIŠKI SISTEM ZA NADZOR IN VODENJE VISOKOREGALNIH SKLADIŠČ

Miha SKUMAVC,
Iskra Avtomatika, Ljubljana, Jugoslavija

UDK: 681.3.014

POVZETEK

V letu 1983 je Iskra Avtomatika TOZD Sistemi skupaj z nekaterimi slovenskimi zastopniki in organizacijami podpisala sporazum o sodelovanju na projektu izgradnje avtomatiziranega visokoregalnega skladiščnega in transportnega sistema avtogum v mestu Togliatti v SZ. Zaradi kompleksnosti sistema smo se odločili za izvedbo upravno-nadzornega sistema v obliki multiračunalniške mreže, ki poleg aparaturne in programske modularnosti zagotavlja tudi ekonomsko-tehnično najučinkoviteje zaokrožen sistem. Z uvedbo lastnega mikroračunalniškega sistema in razvojem sistemske in aplikativne programske opreme smo razvili moderen sistem za nadzor in vodenje skladiščnih in transportnih sistemov.

MULTICOMPUTER SYSTEM FOR THE CONTROL AND SUPERVISION OF HIGH DAY WAREHOUSE

ABSTRACT

In 1983, the Systems Production Division of Iskra Avtomatika together with several other Slovene representatives and firms, signed the cooperation agreement for the construction of an automatized high day warehouse and transportation system Avtogum in Togliatti, Soviet Union. Because of the complexity of the system we decided upon the design of a control and supervision system based on multicomputer network. This network provides not only hardware and software modularity but also an economically and technically most effective self-contained system. With the introduction of our own microcomputer system and with the evolution of system and problem-oriented software, we developed a modern system for the control and supervision of warehouse and transportation systems.

Naloga multiračunalniškega sistema je vodenje in nadzor visokoregalnega skladišča avtogum s pripadajočim vhodnim in izhodnim transportnim sistemom. Zaradi kompleksnosti smo sistem razdelili na dva podsistema, ki zaokrožujeta specifične aktivnosti:

- notranja bilančna lupina obsega ožji skladiščni sistem s:
  - ločenima nivojema za avtomatski vnos in iznos palet,
  - nivo za ročen vnos/iznos palet,
  - visokregalno skladišče z sedmimi hodniki in regali na obeh straneh hodnikov,
  - sedem visokoregalnih dvigal,
- zunanja bilančna lupina obsega periferne transportne sisteme:
  - vhodni transportni sistem,
  - izhodni transportni sistem,
  - transportni sistem za direkten prenos materiala, mimo skladišča, do kupca.

Na tej osnovi smo tudi koncipirali in izgradili modularen multiračunalniški sistem vodenja in nadzora, kot ga predstavlja naslednja oblika hierarhične organiziranosti (slika 1):

- mikroračunalniške končne postaje na visokoregalnih dvigalih,
- podvojena mikroračunalniška centralna postaja,
- miniračunalniški nadzorni center.

Informacijski vodi med končnimi postajami in centralno postajo so izvedeni z VAHLE drsnimi vodi in MKM aparaturnimi moduli na obeh straneh. Komunikacija teče po protokolu "MOSP".

Povezava nadzornega centra in centralne postaje je izvedena po poštni liniji in Data Modemov 2283, na obeh straneh.

Mikroračunalniška končna postaja izvaja samostojno skladiščne postopke na zahtevo centralne postaje. Centralna postaja sporoči v bloku končni postaji:

- vrsto postopka:
  - vskladiščenje,
  - izskladiščenje,
  - klic praznega dvigala,
  - nivo sprejema/oddaja palete,
- koordinate:
  X, Y, Z

Končna postaja opravi zahtevano nalogo, kar obsega:

- prevzem oz. oddajo palete na prevzemno oz. oddajno mesto,
- vožnjo dvigala do zahtevane pozicije. Hitrost prilagaja oddaljenosti od ciljane pozicije. Prav tako zagotavlja istočasnost gibanja v X in Y smeri,
- oddajo oz. sprejem palete z zahtevane pozicije v visokoregalnem hodniku,
- sporočanje koordinat položaja dvigala centralni postaji.

Po izvedeni nalogi končna postaja to sporoči centralni postaji in čaka na naslednjo nalogo. Končna postaja tudi vodi samostojno sliko hodnika s čimer je dana možnost v primeru razpada zveze oz. centralne postaje, enostavnega osveževanja.
Izpad končne postaje ima za posledico generiranje novega ukaza razpoložljivi končni postaji.

Podvojena mikroračunalniška postaja izvaja naslednje obdelave:

- komuniciranje z nadrejenim nadzornim centrom, kar služi:
  - osveževanju slike skladišča,
  - sprejemu zahtev po skladiščnih postopkih,
- sprejema in obdeluje zahteve iz intiligenčne funkcijske tastature,
- na osnovi stranja v skladišču in zahtevanega naloga izdaja blok ukaza izbrani končni postaji,
- vodi samostojno sliko skladišča in jo osvežuje,
- komunicira s končnimi postajami,
- poslužuje pisalnika (odprti in zaprti protokol),
- poslužuje sinoptično sliko na komandnem pultu.

Centralna postaja je aparaturno podvojena. Oba vhodna mikroračunalniška podsistema zajemata istočasno vse informacije in jih samostojno obdelujeta. Izdaja informacij je dovoljena le podsistemu, ki je v stanju aktivnosti. Sistemu v vroči rezervi je izdaja informacij onemogočena aparaturno ali programsko.

Izpad centralne postaje ima za posledico ročno izvajanje skladiščnih postopkov. Ob vzpostavitvi sistema so vsi posegi, iz končnih postaj, prenešeni avtomatsko z dopolnitvijo operaterja na funkcijski tastaturi.

Intiligenčna funkcijska tastatura omogoča dialog operater-sistem. Cilj dialoga je:

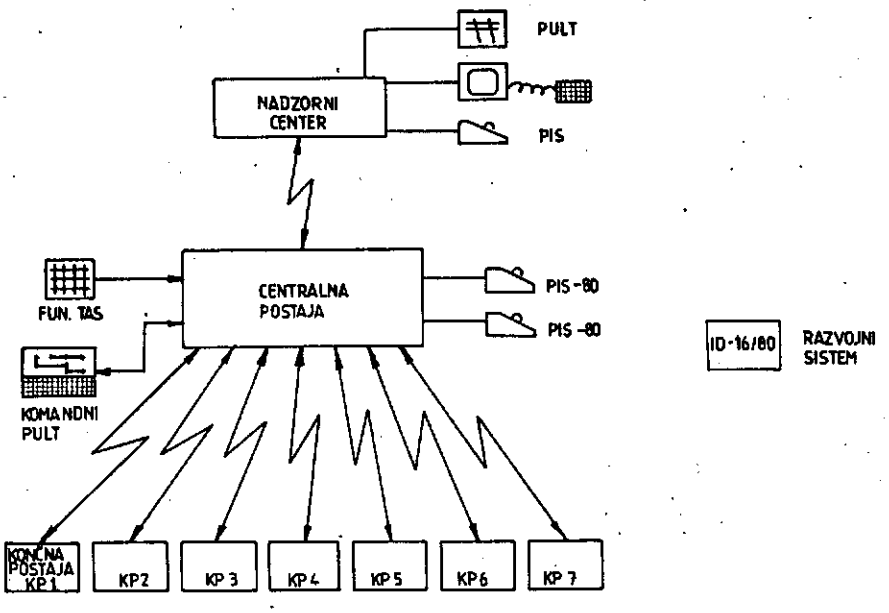
- avtomatsko vskladiščenje/izskladiščenje,
- polavtomatsko vskladiščenje/izskladiščenje,
- izskladiščenje celotne izbrane vrste,
- izskladiščenje celotne izbrane sarže (starost),
- izskladiščenje ene palete izbrane vrste in sarže,
- transport mimo skladišča,
- osveževanje slike skladišča po presoji operaterja,
- izpisi protokolov,
- vnosi datuma in časa.

Centralna postaja omogoča tudi:

- prehode z ene na drugo vhodno transportno linijo,
- avtomatske preklope paketa izdelka v laboratorij,
- štetje in kontrolo števnih grup. Števna grupa predstavlja eno števno mesto. Kontrola števne grupe (3 števci) je izvedena programsko.

Miniračunalniški center služi višenivojskim obdelavam proizvodno-skladiščnega sistema tovarne VAZ Togliatti.

Sistem nadzora in vodenja je uspešno prestal sistemske teste v Metalni v Mariboru.



Slika 1: Sistem za nadzor vodenje VRS

PROGRAMSKA OPREMA MIKRORAČUNALNIŠKEGA SISTEMA ZA NADZOR IN VODENJE VISOKOREGALNEGA SKLADIŠČA

JENKO MILOŠ
ISKRA AVTOMATIKA, LJUBLJANA
JUGOSLAVIJA

UDK: 681.3.014

Mikroračunalniški sistem za nadzor in vodenje visokoregalnega skladišča je zasnovan na mreži enajstih mikroračunalniških enot. Programska oprema je modularna in je deljena na sistemske in aplikativne programske module. Sistemski moduli se različno parametrirani nahajajo v vseh mikroračunalniških enotah. Aplikativni programski moduli so preslikava tehnološkega procesa. Programska oprema je pisana v zbirniku M6800 in višjeprogramskem jeziku MPL. Razvoj in test programske opreme je izvršen na razvojnih sistemih Motorola in Hewlett Packard.

SOFTWARE OF MICROCOMPUTER SYSTEM FOR THE CONTROL AND SUPERVISION OF HIGH DAY WAREHOUSE

The microcomputer system for the control and supervision of high day warehouse consists of a network of eleven microcomputer units. Its software is modularly designed and divided into system and application modules. The system modules of different parameters are incorporated in all microcomputer units. The application modules are a projection of technological process. The software is written in M6800 assembler and in MPL high level programming language. The software is developed and tested on the Motorola and Hewlett Packard systems.

Mikroračunalniški sistem je zasnovan na računalniški mreži, ki zajema:
- končne mikroračunalniške postaje, ki se nahajajo na regalnih dvigalih
- podvojeno centralno mikroračunalniško postajo, ki se nahaja v centru vodenja.

Centralna mikroračunalniška postaja vsebuje dva podsistema:
- mikroračunalniški podsistem za posluževanje funkcijske tastature, zajem informacij iz procesa in krmiljenje sinoptike
- mikroračunalniški podsistem za vodenje in nadzor skladiščnih postopkov in krmiljenje izhodnih enot (printerjev).

Aparaturno opremo končnih postaj in centralne postaje predstavljajo aparaturni moduli teleinformacijskega sistema TI-30. Vsaka končna postaja in podsistem centralne postaje je lokacijsko samostojna enota in vsebuje naslednje sistemske aparaturne module:
- centralno procesno enoto
- delovno-programski pomnilnik
- dajalnik časovnih signalov

in vhodno-izhodne module:
- modul dinamičnih digitalnih vhodov
- modul digitalnih vhodov
- modul digitalnih izhodov
- serijski komunikacijski vmesnik
- modul za modemsko komunikacijo.

Programska oprema vsake samostojne enote je zgrajena modularno in se sestoji iz:
- sistemskih programskih modulov, ki so nujno potrebni za delovanje programske opreme
- aplikativnih programskih modulov, ki izvršujejo funkcije nujno potrebne za izvršitev skladiščnih postopkov.

Modularnost programske opreme omogoča apliciranje programske opreme, ker so funkcije razbite na več samostojnih modulov. Sprememba funkcije ne zahteva velikih posegov v zgradbo celotne programske opreme. Neodvisne programske module povezuje med sabo nadzorni program 'MONMOS'. Vsak modul ima v programskem paketu svojo prioriteto. S pomočjo koordinacijskih števcev določimo število ponavljanj posameznih programskih modulov. Centralna procesorska enota bo dodeljena programskemu modulu na podlagi prejšnih dveh kriterijev in stanja programskega modula.

Monitor MONMOS omogoča startanje programskega modula na več načinov. Normalno je programski modul startan iz drugega modula. Med programskimi moduli se informacije izmenjujejo preko obtočnih pomnilnikov. Celotna sinhronizacija procesorja z zunanjim svetom sloni na prekinitvenih signalih. Zmogljivost sistema se s tem zelo poveča. Informacije prevzame le takrat, ko so na voljo. Digitalne informacije v končnih postajah prevzame le takrat, ko spremenijo stanje. Pri maksimalni hitrosti regalnega dvigala informaciji "začetek čitanja" in "konec čitanja" spremenita stanje v času 6 ms. Pri cikličnem pregledovanju vhodnih informacij sistem ne bi mogel zaznati vseh sprememb stanj. Monitor MONMOS opravi le najnujnejšo obdelavo prekinitev. Nadalnjo obdelavo prekinitev izvrši sistemski programski modul 'RAZINT', ki ima v programskem paketu najvišjo prioriteto. Za vsako vrsto prekinitev izvrši specifične obdelave.

V tej mikroračunalniški mreži se izvajajo naslednje komunikacije:

- končna postaja - centralna postaja
- centralna postaja - uporabnikov miniračunalnik

Komuniciranje teče po protokolu ´MOSP´ in ga izvaja sistemski programski modul ´KOMUN´. Hkrati krmili tudi dva printerja, ki se nahajata v centru vodenja. Programski modul KOMUN omogoča paralelno delovanje na vseh kanalih.
V prenos sporočila je zajeto:
- sprejem sporočila od aplikacijskega modula
- prenos sporočila po liniji
- izdaja sporočila aplikacijskem modulu.

Programska oprema končne postaje vsebuje poleg sistemskih programskih modulov še aplikacijske programske module, ki omogočajo izvršitev naslednjih nalog:
- vodenje visokoregalnega dvigala po hodniku
- nalaganje palete iz regalnega mesta in sprejemne mize
- odlaganje palete v regalno mesto in na oddajno mizo
- spremljanje pomika dvigala pri ročnem vodenju
- diagnostika dvigala.

Vsi dajalci signalov, locirani na dvigalu, so vezani na aparaturni modul dinamičnih digitalnih vhodov. Dajalca binarnih vrednost X in Y pozicij sta vezana na modul digitalnih vhodov. Izvajanje vseh zgoraj navedenih nalog za končno postajo temelji na zajemu in obdelavi prekinitve. Dajalci prekinitev so nekateri vhodno-izhodni aparaturni moduli npr. serijski komunikacijski vmesnik, modul dinamičnih digitalnih vhodov. Prvo razpoznavanje prekinitve določa tip dajalca prekinitve. Nadalnjo obdelavo z dodajanjem časa nastopa prekinitve vrši modul "RAZINT" in preda informacijo modulu ´OBDEL´, modulu za obdelavo vseh vhodnih informacij. Vsak signal ima možnost startanja enega ali več aplikacijskih modulov. Ob startu dobi aplikacijski modul tudi informacijo, ki vsebuje podatke:
- naziv signala
- čas nastopa signala
- stanje signala.

Povprečni čas od nastopa prekinitve do starta aplikacijskega modula traja 9ms. Zaradi tehnološke zahteve, ki dovoljuje največji dovoljeni čas 6ms, pa sta signala o nastopu in koncu X in Y pozicije optimizirana. Aplikacijski moduli so obveščeni o spremembi stanja predhodnih signalov že po 4ms. Modul "VSKL" izvrši vskladiščenje ene palete od sprejema iz sprejemne mize in vnosa v regalno mesto. Modul ´IZSKL´ izvrši izskladiščenje ene palete od iznosa iz regalnega mesta do izdaje na oddajno mizo. Oba modula predstavljata niz klicanja procedur. Ti klici so medsebojno logično povezani glede na vrsto naloge. Procedure so skupne in jih uporabljata oba modula. Procedura predstavlja programski zapis tehnološke zahteve. Zbirka procedur opravlja naslednje tehnološke zahteve:
- pomik regalnega dvigala v X smeri
- pomik regalnega dvigala v Y smeri
- ugotavljanje doseganja zahtevane pozicije
- cikel teleskopa.

Alarmna stanja regalnega dvigala obdeluje modul ´ALARM´. Naziv alarmnega signala, ki ga modul prejme od modula ´OBDEL´ spremeni v kodo. To kodo in čas nastopa signala pošlje preko modula ´ZVEZA´ v center vodenja. Pri ročnem vodenju dvigala je tehnološka zahteva po spremljanju dvigala in pošiljanju pozicije v center vodenja. Pri ročnem vodenju so vsi signali namenjeni v modula ´VSKL´ in ´IZSKL´ usmerjeni v modul ´ALARM´. Tako je temu modulu omogočeno sledenje dvigala. Vse izhodne komunikacije s centrom vodenja potekajo preko modula ´ZVEZA´. Vsaka informacija se predhodno shrani v obtočni pomnilnik modula ´ZVEZA´ in se prenese v center samo ob pravilnem delovanju komunikacije. S tako programsko rešitvijo je omogočeno tudi ročno vodenje dvigala kljub izpadu zveze končna postaja - center vodenja.

Programska oprema podsistema za vodenje in nadzor vsebuje poleg sistemskih programskih modulov še aplikacijske programske module, ki so grupirani. Moduli znotraj grupe imajo sosednje prioritete, grupe pa imajo med sabo večje prioritetne razlike. Najpomembnejši je modul za nadzor dvojnoračunalniškega sistema. Dinamično osvežuje stanje sosednjih mikroračunalniških enot. Programsko je deljen na dva dela, ki sta preslikavi stanja mikroračunalniške enote. Prehod iz enega stanja v drugega pogojujejo izpad mikroračunalniške enote ali izpolnitev programskih pogojev za preklop. Grupa modulov za sprejem sporočil iz podsistema za posluževanje funkcijske tastature preverja logično pravilnost vnešenega posluževanja. Informacijo o posluževanju preoblikuje v zahtevano obliko za grupo modulov za protokoliranje in jo pošlje tej grupi. Glede na vejo posluževanja na funkcijski tastaturi in njej pripadajoči kodi ugotovi modul v grupi za nadzor, ki bo prevzel posluževanje. Sprejem sporočil iz končnih postaj, ki vsebujejo kodo alarma, ravno tako poteka preko te grupe modulov. V tej grupi se sporočilo preoblikuje v zahtevano obliko za grupo modulov za protokoliranje. Grupa modulov za nadzor vsebuje število aplikacijskih modulov, ki je enako številu samostojnih tehnoloških operacij v skladišču. Razpolagajo z skupno programsko sliko skladišča. V tej sliki je vsako regalno mesto opisano z tremi zlogi, ki vsebujejo informacije o regalnem mestu za vse tehnološke operacije. Slika je vpisno-čitalna. Grupa vsebuje tudi modul za sinhronizacijo med moduli pri navideznem paralelnem delovanju večih tehnoloških operacij. Grupa modulov za protokoliranje vsebuje modul za odprti protokol in modul za zaprti protokol. Za vsa sporočila namenjena za odprti protokol modul poišče pripadajoče tekste v banki tekstov in preko sistemskega modula "KOMUN" izpiše sporočilo na printer. Modul za zaprti protokol sestavi formular zahteva niz strani investitorja in je tekstovno in oblikovno vedno enak, spremenljive pa so številčne vrednosti parametrov.

Programska oprema podsistema za posluževanje vsebuje potrebne sistemske module in aplikacijske programske module. Iz funkcijske tastature so vsi signali vezani na aparaturne module dinamičnih digitalnih vhodov. Postopek prenosa informacije ob nastopu prekinitve je enak kot pri zgoraj opisani končni postaji. Informacijo o prekinitvi sprejema iz modula ´OBDEL´ modul ´FUNTAS´. Ta modul preverja sintaktično pravilnost posluževanja in tudi krmili sinoptiko namenjeno funkcijski tastaturi. Sintaktično pravilno vnešeno posluževanje prenese preko komunikacijskega protokola grupi modulov za sprejem sporočil. Modul za krmiljenje sinoptike ima možnost krmiljenja ostale sinoptike na dva načina. Po prvem načinu ob zahtevi postavi lučko v stanje ´trajno goreče´. Po drugem načinu po postavi lučko v stanje ´utripajoče´. Drugi način je izveden z dajalnikom časovnih signalov. Ta starta pregledovanje vpisno-čitalne liste, ki se dinamično popravlja. Lista vsebuje vse nazive lučk, ki so v stanju ´utripajoče´.

MODEL RAČUNALNIŠKEGA RAZPOREJANJA CESTNIH TOVORNIH VOZIL

LJUBO GULIČ INTEREUROPA KOPER JUGOSLAVIJA

UDK: 681.3.02

Razporejanje cestnih tovornih vozil je razdeljeno na sledeče faze: iskanje vozil po nahajališčih, določanje vozila glede na tovor in omejitve vzdrževanja ter izbira najprimernejše poti. Cilj je določitev najprimernejšega vozila in poti za naročeni prevoz. Zamišljeni model zahteva predhodno zbrane podatke o vozilu in relacijah po katerih naj bi se vršil prevoz. Podatki o vozilu obsegajo njegovo trenutno nahajališče in stanje z vidika vzdrževanja. Najprimernejšo relacijo med dvema točkama določimo z uporabo dinamičnega programiranja kot posebne panoge uporabne matematike. Z uporabo vseh zbranih in točnih podatkov določa računalniški model možna vozila na optimalni relaciji. Končna odločitev je prepuščena razporejevalcu. Obstaja možnost programiranega odločanja kot višja razvojna stopnja obravnavanega modela.

THE MODEL OF THE COMPUTERIZED DISPOSITION OF ROAD FREIGHT VEHICLES

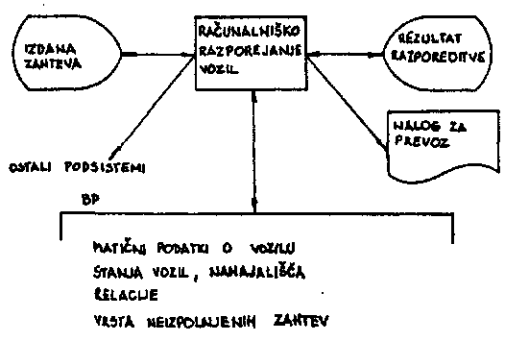
The disposition of the road freight vehicles is devided into the following phases: search for vehicles at the depots, appointment of vehicles with regard to the cargo and to the upkeep limit, choise of the most suitable route. The purpose is to appoint the most suitable vehicle and route for the conveyance ordered. The model so conceived requires some previously collected particulars about the vehicle as well as about the routes. The data concerning the vehicle include its actual condition and the depot as far as the upkeep is concerned. The most suitable route between two points is fixed by a dynamic programming as a special branch of applicable mathematics. The computerś model determines bothe the vehicle and the best route on the basis of all collected information. The final decision is taken by the person who is in charge of the vehicles appointment. There also exists the possibility of a programmed decision-making as a higher development of the model concerned.

I. UVOD

V dobi vse bolj zaostrene energetske krize so stroški cestnega tovornega prevoza (izvajanje prevoza troši občutne količine energije) vedno bolj podvrženi njihovemu proučevanju. Stroški fizične distribucuje (katere del je tudi prevoz blaga) so postali pomemben stroškovni element, ker so proizvodni in ostali stroški znižanmi Cilj je zmanjšati prevozne stroške. Pravilne odločitve omogočajo to zmanjševanje, zato mora odločevalec razpolagati s primernimi informacijami. Samo ustrezen informacijski sistem (IS) lahko oblikuje take informacije. Računalniški IS (RIS) zagotavlja izvrševanje razporejanja vozil ob nastanku zahteve po prevozu ter omogoča posredovanje odločevalca z in interakcijo v tekočem času. Zamišljeni model razporejanja zahteva predhodno zbrane podatke o stanju vozil (nahajališče, omejitve vzdrževanja) in o relacijah po katerih naj bi se vršili prevozi.

2. OPIS MODELA

RIS za razporejanje vozil oblikuje potrebne informacije za odločitev: katero vozilo po kateri poti po zahtevi za prevoz med določeno začetno in končno točko. Začetna vzpodbuda je dana iz poslovne funkcije prodaje tovornega prostora. Vsebuje vse potrebne podatke in informacije za razporeditev (mesto natovarjanja in raztovarjanja, vrsta tovora, naročnik). Slika 1 prikazuje zasnovo obravnavanega RIS-a.



Slika 1. RIS za razporejanje vozil

Prispele zahteve se shranijo v vrsto, ki se z algoritmom prazni v določenih časovnih presledkih. Neizpolnjene zahteve se ponovno shrani v vrsto ali vrne prodajni funkciji.
Podsistemi, ki oblikujejo podatke in informacije za razporejanje pred prispelo zahtevo za prevoz so:
- podsistem spremljanja vozil (gibanje in trenutno nahajališče),
- podsistem vzdrževanja vozil,
- podsistem določanja optimalne poti.

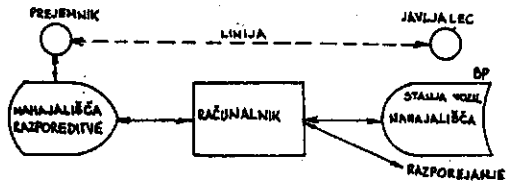
Podali bomo kratek opis posameznega podsistema zaradi tesne povezanosti s podsistemom razporejanja vozil. Vsi navedeni podsistemi so elementi obravnavanega RIS-a, ki je del celotnega RIS organizacije za opravljanje cestnega prevoza blaga.

2.1. Spremljanje vozil

Podsistem spremljanja vsebuje tri osnovne funkcije:
- javljanje voznikov o nahajališču,
- vnos vseh potrebnih podatkov (dodatno opremljanje zapisa) in
- shranjevanje podatkov v bazo podatkov (BP).

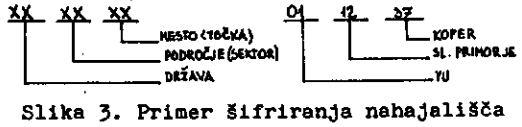
Komunikacijski model javljanja nahajališča vsebuje naslednje elemente: javljalec, veza, sprejemnik in motnje. Za javljanje se uporablja telefon, teleprinter ali obstoječe linije za povezavo poslovnih enot s centralnim računalnikom. Vse zveze so usmerjene na eno mesto. Od tu se sporočila vnašajo preko ekranskega terminala v BP. Isti kanal se uporablja za sporočanje voznikom o naslednjem mestu razporeditve. Težave nastopijo, ko je potrebno istočasno javljati položaj in dajati navodila o novi razporeditvi. Odpravi se jih z uskladitvijo teh sporočil in določanjem prioritete posameznih vrst sporočil. Sprejete podatke se opremi in dopolni pred vnašanjem v BP. Javljalec sporoči sledeče:
- identifikacijo vozila,
- trenutno in predvideno naslednje nahajališče in časovne elemente ter
- posebnosti ali nepredvidene omejitve.

Vsi ostali podatki so podani z zadnjo razporeditvijo tega vozila. Algoritem za vnos vsebuje funkcije vnosa podatkov, njihove vsebinske in formalne kontrole ter zapisa v BP. Slika 2 prikazuje izvedbo opisanega podsistema.



Slika 2. Podsistem za spremljanje vozil

Identifikacija nahajališča je lahko različno opredeljen pojem. Predstavlja teritorijalni obseg v določeni državi (področje), večjo industrijsko središče, pogosto prehodno mesto ali drugače določeno točko. Za računalniško vodenje nahajališč in razporejanje vozil mora biti ta pojem primerno šifriran. Dolžina in oblika šifre je odvisna od zahtevane natančnosti nahajališča. Na sliki 3 je prikazan primer take šifre.

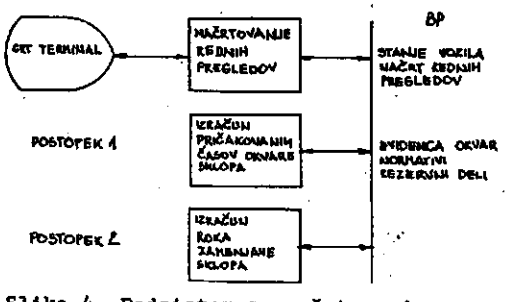


Slika 3. Primer šifriranja nahajališča

Spremljanje in uvajanje novih nahajališč je izvedeno izkustveno s posredovanjem samih voznikov pri zaključevanju prevoza. Vozniki so pomemben vir podatkov za obravnavani RIS.
V trenutku javljanja lahko obstajata dve možnosti: dana je že naslednja razporeditev ali pa te še ni. V prvem primeru je čas pozivanja in njegovi stroški minimalen. V drugem primeru so potrebni večkratni pozivi (primerni časovni intervali), kar povečuje stroške javljanja. Celotna opravljena pot mora biti enaka stanju javljanja o tej poti. V primeru odstopanja je vzrok izostanek sporočanja voznika o nahajališču. Število dnevnih interakcij v obravnavanem podsistemu je odvisno od števila vozil, razporeditve in zasedenosti (povpraševanja po tovornem prostoru). Trenutna stanja vozil v BP morajo biti zelo približana resničnemu stanju, sicer je računalniško razporejanje vozil neuporabno. Točnost je zelo odvisna od discipline javljanja (voznik, prejemnik podatkov).

2.2. Vzdrževanje vozil

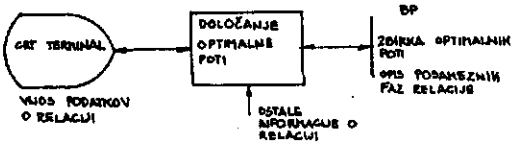
Obravnavali bomo samo tisti del podsistema, ki oblikuje informacije o razpoložljivosti vozila z vidika vzdrževanja. Ta del zajema načrtovanje rednih servisnih pregledov, večjih popravil in zamenjave sklopov ter vsklajevanje zahtev vzdrževanja in eksploatacije vozil.
Določanje rednih servisnih pregledov je odvisno od normativov in stanja vozila (km, ure obratovanja). Vsako odstopanje od rednega pregleda povzroči večje težave pri kasnejšem popravilu. Načrtovanje obsega določanje datuma in ure prihoda v delavnico ter pripravo (kadri in rezervni deli). Seveda so osnova načrtovanju prevoženi kilometri ali ure obratovanja. To načrtovanje ni zahtevno, ker je s proizvajalčevimi predpisanimi roki ali km določeno izhodišče in je samo od odločevalca (vzdrževalca) odvisno ali bo upošteval sprejeto informacijo o zahtevi po rednem pregledu vozila. Rezultati načrtovanja so shranjeni v zapis o celotnem stanju vozila.
Načrtovanje večjih popravil in zamenjave sklopov je težje zaradi nepredvidenih časov nastopa okvare. Sodoben način vzdrževanja cestnih tovornih vozil je usmerjen k zamenjavi sklopov ter kasnejšemu popravilu zamenjanega sklopa. Tako je vozilo malo časa izven uporabe. Osnova temu načrtovanju je evidenca okvar v preteklih obdobjih, evidenca vzrokov in tehnična navodila (norme) proizvajalca. S pomočjo teh evidenc in matematičnih metod (stohastika) lahko določimo pričakovani čas nastanka okvare. Te čase določimo za vsak sklop posebej ter postanejo del normativov za te sklope. Občasno ponovimo postopek ugotavljanja teh časov z upoštevanjem novih podatkov o nastalih okvarah. Slika 4 podaja izvedbo podsistema za načrtovanje pregledov in zamenjav. Postopek 2 na sliki 4 ponavljamo v istih časovnih presledkih kot načrtovanje rednih pregledov. Rezultati so shranjeni v celotnem zapisu o stanju vozila.



Slika 4. Podsistem za načrtovanje popravil

2.3. Določanje optimalne poti

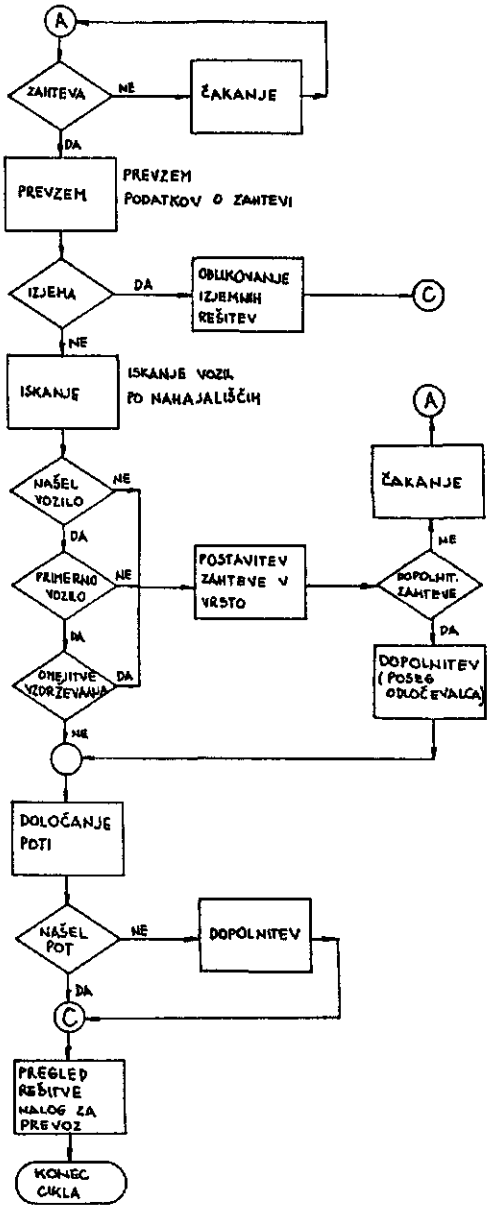
Opravljanje prevozov na določeni poti opišemo kot večfazni proces. Posamezne faze predstavljajo odseke med začetno in končno točko izbrane poti. Izbiro optimalne poti izvedemo z uporabo dinamičnega programiranja kot posebne panoge uporabne matematike (metoda diskretnega dinamičnega programiranja). Proces izvajanja prevoza poteka ob spreminjanju neke enodimenzionalne spremenljivke (čas, razdalja, cena, vrednost). Računalniško določanje optimalne poti zahteva določitev osnovnih izhodiščnih elementov (izvedba algoritma ni predmet tega dela). Ti elementi so: vrsta kriterija, ekstrem in velikost cestne mreže, ki je zajeta med začetno in končno točko relacije za katero iščemo optimalno pot. Rezultat je podan z opisom celotne relacije (vmesne točke) ter je shranjen v BP za nadaljno uporabo pri razporejanju. Iskanje optimalne poti se izvaja ločeno v določenih časovnih presledkih (ko zberemo vse podatke o določeni relaciji). Tudi v tem primeru so vozniki pomemben vir podatkov poleg avtokart in informacij o stanju cest, predorov in vzponov. Dodajanje rezultatov v BP stalno povečuje zbirko optimalnih poti. Občasno se že obstoječe izračunane poti ponovno obdela s podatki o novem stanju kriterijev. Vse relacije za katere ni še določena optimalna pot (določi jo razporejevalec) so ravno tako shranjene v zbirki relacij. Zahteva po določanju optimalne poti z več kriteriji (različne enote mer) nas privede do obveznega pretvarjanja posameznih vrst kriterijev na skupno enoto. Računalniško določanje je težko izvedljivo, ker zahteva poseg odločevalca zaradi določanja ponderja ali pretvarjanja. Pretvorba na skupno enoto ni vedno enostavna zaradi različnih pomenov kriterijev (čas, ovira na cesti, cestnina). Najprimerneje je vse kriterije pretvoriti na dve enoti in sicer na časovne in vrednostne. Izkustveno razmerje popači absolutne lastnosti. Možna rešitev je v ločenem določanju optimalne poti za vsak kriterij posebej. Slika 5 prikazuje izvedbo opisanega podsistema.



Slika 5. Podsistem določanja optimalne poti

2.4. Razporejanje vozil

Podsistem RIS-a za razporejenje vozil zahteva izvajanje predhodno opisanih podsistemov. V nasprotnem primeru je rezultat razporejanja neuporaben. Algoritem oblikovanja odločitve o razporeditvi je prilagojen sprotnim zahtevam. Odločevalec ima možnost stalnega posega v njegov potek. Dopolnjuje zahtevo po prevozu in dodaja manjkajoče delne ali nepravilne rešitve zaradi napačnih podatkov ali njihovega izostanka. Potek izbire vozila je podan na sliki 6. Prodajna funkcija oblikuje zahtevo in jo preda v vrsto. Vsebuje podatke o naročniku, tovoru in posebnostih. Odločevalec prazni vrsto po svoji presoji. Sledi iskanje primernega vozila (nahajališče, tovor in omejitve vzdrževanja). Algoritem izbere vsa vozila, ki so prijavljena na danem področju. Razvrščena so od najbližjega do še sprejemljivo oddaljenega od točke začetka prevoza. V primeru, da ne obstaja vozilo na tem področju posreduje odločevalec in določi status zahteve (počaka, da se pojavi vozilo na tem področju ali določi drugo vozilo).



Slika 6. Algoritem razporejanja vozil

Naslednji korak algoritma preverja lastnosti vozil in tovora. Vrsta in količina določata potrebne lastnosti vozila za ta prevoz. Temeljna lastnost je nosilnost, ostale pa so: prostornina tovornega prostora, poseben ustroj podvozja, hitrost. Tako izbrana vozila se preverja z omejitvami vzdrževanja. Dolžina in čas trajanja predvidenega prevoza lahko prekoračita določene zahteve za redni pregled ali zamenjavo sklopa. V primeru, da je izbranih več vozil lahko odločevalec izbira na osnovi danih kriterijev (najbližje vozilo, ocena voznika, število voženj) ali po svoji presoji. Ravno tako lahko uvrsti zahtevo v vrsto ali sam določa vozilo, če je rezultat iskanja vozil negativen. Po določitvi vozila sledi določanje poti

iz zbirke optimalnih poti. Zahteva za prevoz vsebuje začetno in končno točko prevoza. Algoritem ugotavlja prisotnost te relacije v zbirki optimalnih poti. V primeru, da ne obstaja željena relacija, odločevalec sam določi pot prevoza. S tem je podana rešitev razporeditve vozila. Odločevalec potrdi ali dopolni rešitev in jo preko podsistema spremljanja vozil posreduje vozniku izbranega vozila. Izvajanje take razporeditve zahteva veliko vhodno-izhodnih operacij (diski, linije za ekranski terminal in povezava z ostalimi podsistemi) in kratke čase odziva.Razporejenost vozil in njihovo število so osnova za določanje računalniških kapacitet. Poseben primer (izjema) nastopi v primeru, ko odločevalec sam določi vozilo in se ne izvajajo vsi koraki algoritma. Opisani podsistem lahko nudi prodajnemu podsistemu informacije o nerazporejenih vozilih. Tako je povečana izkoriščenost vozil (išče se tovor za ta vozila pri možnih naročnikih).

3. MOŽNOSTI PROGRAMIRANEGA ODLOČANJA

Obravnavani model razporejanja prepušča končno odločitev o izbiri vozila in poti razporejevalcu, vendar vsebuje nekaj elementov za programirano odločanje. Ti elementi so: cilji, spremenljivke katere lahko nadziramo ali ne ter pravila odločanja v obravnavanem modelu. Zavedamo se, da obstajajo realne možnosti za uvedbo programiranega odločanja na operativnem nivoju procesa izvajanja prevoza. Cilj na tem nivoju je razporeditev vozila za določeni tovor na določeni poti. Spremenljivke so sledeče: vrsta tovora z vsemi podanimi lastnostmi in pogoji, mesto in čas prevzema in dostave blaga, tehnično stanje in lastnosti vozila ter nahajališče. Izvajalec lahko delno vpliva na tehnično stanje in čas dostave tovora. Ostale spremenljivke so dane z naročilom prevoza. Pravilo odločanja se glasi: določi tisto vozilo za naročeni prevoz, ki je najbližje zahtevanemu mestu prevzema blaga, po svojih tehničnih lastnostih lahko prepelje tovor po optimalni poti in nima nobenih zadržkov zaradi vzdrževanja. Navedeni elementi programiranega odločanja so podobni elementom algoritma za razporejanje vozil. Pravilo odločanja je izpeljano za uspešne izide posameznih stopenj preverjanja stanja. Težava nastopi, ko določena stopnja preverjanja ne zagotovi uspešnega izida. Razširiti moramo pravilo odločanja za tak primer in prilagoditi parametre (obseg). Ti so vsebovani v BP. Vsaka nadaljna stopnja razširjanja povzroči odmik od optimalne rešitve. Rezultat ni optimalna odločitev je pa edina možna. Prevelik odmik povzroči zavrnitev zahteve po prevozu ali čakanje te zahteve v vrsti za razporejanje. Določitev vseh parametrov je v tesni povezavi z ekonomskimi učinki rešitve razporejanja. Stalno prilagajanje parametrov dejanskemu stanju omogoči učinkovito programirano odločanje pri razporejanju vozil.

4. ZAKLJUČEK

Učinek prevoza, kot osnovno merilo vrednosti procesa prevoza, je pogojen z kakovostjo izvedbe razporejanja vozil (ni edini pogoj). Nakazali smo možne vsebinske rešitve opisanega RIS za razporejanje vozil. Ostali elementi (računalniška oprema, organiziranost DO) so odvisni od organizacije za prevoz blaga. Zavedamo se, da je oblikovanje RIS obsežno delo, ki zahteva skupinsko delo strokovnjakov iz vseh področij poslovanja DO ter ustrezno znanje in opremo. Upamo, da je opisani model dovolj dobra osnova za pristop k oblikovanju resničnega RIS-a za razporejanje vozil in da bo uspešna izvedba potrdila naša razmišljanja.

Literatura:

Burch and Strater....Information Systems: Theory and Practice, Wiley/Hamilton Publ. Santa Barbara 1974

A. Vadnal.....Diskretno dinamično programiranje, DZS, Ljubljana 1976

J. Grad.....Določitev optimalne poti z metodo diskretnega dinamičnega programiranja, Ekonomska revija, Ljublj. št.1-2/1979

M. Marunica.....Tehnološko-transportno opisivanje itinerara u cestovnom robnom transp. u funkciji upravljanja, Suvremeni promet,1982/2

L. Gulič.....Računalniški informacijs ki sistem v organizaciji za cestni prevoz blaga, magistrsko delo, EF Lj, 1984

RAČUNALNIŠKO PODPRTO NAČRTOVANJE PULZNIH USMERNIKOV

Bojan Alatič, Karel Jezernik
TEHNIŠKA FAKULTETA MARIBOR

UDK: 681.3.06

Povzetek - V referatu je opisan program PASEP za analizo in sintezo pulznih presmernikov. Podane so osnove modeliranja presmernikov v prostoru stanj in razvite prenosne funkcije. Na primeru tranzistorskega mostičnega usmernika je prikazan postopek simulacije. Ta je bila izvajana na mikroračunalniku VAX 11 s pomočjo simulacijskega programa PADSIM.

COMPUTER - AIDED DESIGN OF SWITCH MODE POWER SUPPLYES

Summary- In the paper is described software packet PASEP for analysis and synthesis of switch mode converters. Given are methods for evaluation of modells and transfer functions of converters.On the example of bridge type switch mode power supply is shown proceeding of simulation. The simulation was executed with simulation programm PADSIM on minicomputer VAX 11.

1. UVOD

Na vseh področjih elektronike je v zadnjih letih prišlo do naglega razvoja komponent. Z razvojem polprevodnikov, še posebej hitrih visokonapetostnih diod in tranzistorjev, se je na področju močnostne elektronike odprlo zelo široko aplikacijsko področje - enosmerni presmerniki in pulzni napajalniki.

Načrtovanje pulznih napajalnikov je lahko zelo zamudno delo. Izračune elementov pretvornikov večkrat ponavljamo, vrednosti elementov pa preverjamo ob delovanju pretvornika. Taka pot je dostikrat neuspešna in ne vodi do optimalnih rezultatov. Računalniško podprta analiza, sinteza in simulacija je zato edina smiselna rešitev tega problema. Obsega načrtovanje enosmernih presmernikov, vhodnega (mrežnega) usmernika s filtrom, izhodnega usmernika s filtrom in načrtovanje PŠM s prožilnimi stopnjami za tranzistorje.

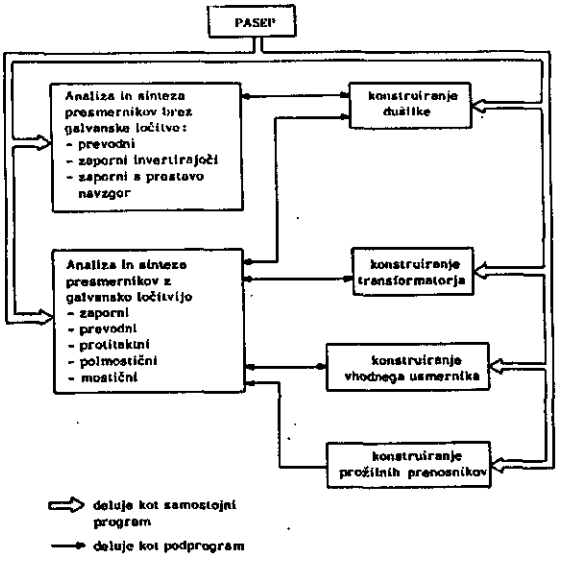
Analizo in sintezo v stacionarnih pogojih izvajamo s programom PASEP. Analizo dinamičnih pojavov v napajalnikih izvajamo na podlagi znanih modelov s prirejenimi simulacijskimi programi.

2. PROGRAM PASEP

Program PASEP je namenjen za analizo in sintezo enosmernih presmernikov. Izračunavanje elementov presmernikov je zamudno in nepregledno, večkrat ga je potrebno ponoviti, zato je na dlani, da za takšno delo uporabimo računalnik. Ker razvojni inženir največkrat nima na razpolago velikega računalnika, osebni in hišni računalniki pa postajajo vedno bolj razširjeni in dostopni, smo razvili program PASEP v programskem jeziku BASIC, ki je na µR najbolj razširjen.

Program PASEP je razdeljen na 12 podprogramov, ki sami zase predstavljajo celoto, lahko pa se med seboj dopolnjujejo. Zgradbo programa kaže sl. 1.

Kot je s slike razvidno, lahko izvajamo sintezo in analizo



Sl. 1: Zgradba programa PASEP

pretvornikov brez in z galvansko ločitvijo, opravimo konstruiranje dušilk in transformatorjev, vhodnih in izhodnih usmernikov in filtrov ter prožilnih stopenj za tranzistorje. Delo z računalnikom je interaktivno. V glavnem meniju izberemo želeni podprogram, ga izvršimo, sam podprogram pa nas vodi v druge podprograme ali pa nazaj v glavni meni.

Vnos podatkov poteka preko tipkovnice, izpis rezultatov pa je na zaslonu ali na tiskalniku.

Podprogrami so razdeljeni na sintizo in analizo. Sinteza

na podlagi zahtevanih vhodno-izhodnih parametrov presmernika izračuna vrednosti elementov L in C, opravi konstruiranje dušilke in transformatorja, izračuna maksimalne in efektivne vrednosti tokov, stikalne čase in poda potrebne parametre polprevodnikov. Analiza na podlagi znanih vrednosti pasivnih elementov opravi izračun napetosti in tokov presmernika, stikalnih časov ter valovitosti napetosti in tokov. V izračunih so lahko upoštevane parazitne napetosti polprevodnikov, ki jih podamo pred pričetkom izvajanja programov [1, 2, 3, 4].

Podprograma za konstruiranje dušilk in transformatorjev na podlagi zahtevanih induktivnosti, tokov in magnetnih gostot feritnih materialov izračunata potrebni ploščinski produkt magnetnega jedra za dušilko ali transformator. Na osnovi potrebnega ploščinskega produkta (velikosti) jedra se opravi izbor iz tabele petih družin feritnih jeder (lončki FL, U jedra FU, RM jedra, EE in EC jedra). Upoštevaje magnetne in ostale parametre izbranega jedra se opravi izračun števila ovojev, potrebnih presekov žic, zračne reže, izračunajo pa se tudi izgube.

Primer izpisa poteka konstruiranja dušilke je prikazan na sliki 2.

```
PODPROGRAM ZA KONSTRUIRANJE
        DUSILKE
B - magnetna gostota 0.3 T
J - tokovna gostota =5 A/mm2
k - izkoristek navitalneva preseka =0.5

Potrebni PloscinskiProdukt jedra AP (= AwAe) =0.15418333 cm2


TABELA FERITNIH LONCKOV

TIP             Ap          Ae          Aw          lsr
                (cm4)       (cm2)       (cm2)       (cm)

FL0905          .0028       0.1         .028        1.85
FL1107          .0067       0.16        .042        2.2
FL1408          .016        0.19        .084        2.8
FL1811          .056        0.35        0.16        3.56
FL2213          0.117       0.5         0.234       4.4
FL2616          0.2368      0.74        0.32        5.2
FL3019          0.5376      1.12        0.48        6
FL3622          1.0899      1.73        0.63        7.3
FL4229          3.81        2.65        1.438       8.38
FL4528          3.306       2.3         1.438       0
FL4728          3.588       3.12        1.15        9.3

IZBRANO JEDRO : FL2616
Ap =0.74 cm2
Aw =0.32 cm2
lsr =5.2 cm

Karakteristicni parametri
dusilke so :

stevilo ovojev      N=5
Presek zice         Sz=0.29(mm2)
izgube v bakru      Pcu=0.295(W)
```

Sl. 2: Računalniški izpis poteka konstruiranja dušilke

3. MODELIRANJE PULZNIH NAPAJALNIKOV S POVPREČENJEM V PROSTORU STANJ

Pri modeliranju pulznih napajalnikov smo do sedaj poznali dve poti - analizo v prostoru stanj in modeliranje nadomestnih vezij [1]. Dobre lastnosti obeh je združila metoda povprečenja v prostoru stanj, saj na podlagi opisa nadomestnih vezav za različna stikalna stanja omogoča razvoj modela v prostoru stanj ali pa razvoj modela s povprečenjem nadomestnih vezij. S tem se opisi pretvornikov poenotijo in veljajo enaki modeli za različne tipe pretvornikov, spremeniti je potrebno le koeficiente posameznih elementov modela [1, 2, 5].

Osnova pretvorbe enosmerne energije je v preklapljanju linearnih vezij, ki so sestavljena iz induktivnih in kapacitivnih elementov. To preklapljanje je izvedeno s stikalnimi elementi - tranzistorji in diodami. Stanje vezja pri različnih položajih (stanjih) stikalnih elementov lahko opišemo z enačbami stanja. Pri izboru spremenljivk stanja ni posebnih predpisov, običajno pa vzamemo kot spremenljivki stanja tok dušilke in napetost kondenzatorja. Katerikoli presmernik, ki deluje s kontinuiranim prevajanjem, lahko predstavimo z enačbami spremenljivk stanja oblike

za interval tvk: $\dot{\underline{x}} = \underline{A}_1 \cdot \underline{x} + \underline{b}_1 \cdot u_i$, $y_1 = \underline{c}_1^T \cdot \underline{x}$

za interval tiz: $\dot{\underline{x}} = \underline{A}_2 \cdot \underline{x} + \underline{b}_2 \cdot u_i$, $y_2 = \underline{c}_2^T \cdot \underline{x}$ (1)

pri čemer velja T = tvk + tiz

Da dobimo povprečeni model, moramo izraza za $\dot{\underline{x}}$ v enačbi (1) množiti z vklopnim razmerjem in ju sešteti. Dobimo enačbi

$$\dot{\underline{x}} = D(\underline{A}_1\underline{x} + \underline{b}_1 \cdot u_i) + D'(\underline{A}_2\underline{x} + \underline{b}_2 u_i)$$

$$y = D \cdot y_1 + D' y_2 = (D\underline{c}_1^T + D'\underline{c}_2^T) \cdot \underline{x} \qquad (2)$$

ki ju lahko pišemo v običajnejši obliki

$$\dot{\underline{x}} = (D\underline{A}_1 + D' \cdot \underline{A}_2) \cdot \underline{x} + (D\underline{b}_1 + D'\underline{b}_2) u_i$$

$$y = (D\underline{c}_1^T + D'\underline{c}_2^T) \cdot \underline{x} \qquad (3)$$

pri čemer velja $D = \frac{tvk}{T}$ in $D' = \frac{tizk}{T} = 1 - D$ (4)

Enačba (3) predstavlja osnovni povprečeni model pretvornika v prostoru stanj. Vklopno razmerje D in vzbujalna napetost u_i sta v izrazu (3) konstanti.

Vpliv motenj na sistem je vpliv spremembe vklopnega razmerja D in napajalne napetosti u_i na stanje sistema. Motnje povzročijo spremembo vektorja stanja in izhodne veličine:

$$u_i = U_i + \hat{u}_i \qquad \underline{x} = \underline{X} + \hat{\underline{x}}$$

$$d = D + \hat{d} \qquad y = Y + \hat{y} \qquad (5)$$

Velike črke pomenijo stacionarne vrednosti, male črke s strešico pa spremembe okrog stacionarnega stanja.

Izraz (3) lahko pišemo v splošni obliki

$$\dot{\underline{x}} = \underline{A} \cdot \underline{x} + \underline{b} \cdot u_i$$

$$y = \underline{c}^T \cdot \underline{x} \qquad (6)$$

pri čemer velja

$$\underline{A} = D \cdot \underline{A}_1 + D' \underline{A}_2$$

$$\underline{b} = D\underline{b}_1 + D'\underline{b}_2 \qquad (7)$$

$$\underline{c}^T = D\underline{c}_1^T + D'\underline{c}_2^T$$

Če sedaj izraze (5) vstavimo v (6), dobimo osnovni model v obliki

$$\dot{\underline{x}} = \underbrace{\underline{A}.\underline{X}+\underline{b}.U_i}_{a} + \underbrace{\underline{A}.\hat{\underline{x}}+\underline{b}.u_i}_{b} + \underbrace{[(\underline{A}_1-\underline{A}_2).\underline{X}+(\underline{b}_1-\underline{b}_2)U_i]\,\hat{d}}_{c} + \underbrace{[(\underline{A}_1-\underline{A}_2)\hat{\underline{x}}+(\underline{b}_1-\underline{b}_2)\hat{u}_i]\,\hat{d}}_{d} \quad (8)$$

a - stacionarni (DC) izraz
b - variacija vhodne napetosti
c - variacija prevajalnega razmerja
d - nelinearni člen 2. reda

Izhodna veličina ima obliko:

$$y = Y + \hat{y} = \underline{c}^T.\underline{X} + \underline{c}^T.\hat{\underline{x}} + (\underline{c}_1^T - \underline{c}_2^T)\underline{X}.\hat{d} + (\underline{c}_1^T - \underline{c}_2^T)\hat{\underline{x}}.\hat{d} \quad (9)$$

Z linearizacijo enačb (8) in (9) dobimo končni povprečeni model v prostoru stanj:

stacionarni model

$$\underline{X} = -\underline{A}^{-1}.\underline{b}\,U_i$$

$$Y = \underline{c}^T.\underline{X} = -\underline{c}^T.\underline{A}^{-1}.\underline{b}.U_i \quad (10)$$

dinamični model

$$\dot{\hat{\underline{x}}} = \underline{A}.\hat{\underline{x}}+\underline{b}.\hat{u}_i + (\underline{A}_1-\underline{A}_2).\underline{X}+(\underline{b}_1-\underline{b}_2)U_i\,\hat{d}$$

$$\hat{y} = \underline{c}^T.\hat{\underline{x}} + (\underline{c}_1^T - \underline{c}_2^T)\,\underline{X}\,\hat{d} \quad (11)$$

4. PRENOSNE FUNKCIJE SISTEMA

Pulzni napajalnik običajno deluje kot sistem z zaprto regulacijsko zanko za regulacijo izhodne napetosti (ali toka). Ker je pretvornik s pulzno-širinskim modulatorjem nelinearni podsistem z dvema vhodoma in enim izhodom, je za analizo v zveznem prostoru potrebno razviti zvezne prenosne funkcije. Najpomembnejši sta prenosni funkciji izhodna napetost - vhodna napetost pretvornika in krmilna prenosna funkcija izhodna napetost - krmilna napetost PŠM.

S transformacijo dinamičnega povprečenega modela v slikovni prostor dobimo izraz

$$\hat{\underline{x}}(p) = (p\underline{I}-\underline{A})^{-1}.\underline{k}.\hat{d}(p) + (p\underline{I}-\underline{A})^{-1}.\underline{b}.u_i(p) \quad (12)$$

Od tod dobimo želene prenosne funkcije

$$\frac{\hat{y}}{\hat{d}}(p) = \underline{c}^T(p\underline{I}-\underline{A})^{-1}.\underline{k} + (\underline{c}_1^T - \underline{c}_2^T).\underline{X} \quad (13)$$

$$\frac{\hat{y}}{\hat{u}_i}(p) = \underline{c}^T(p\underline{I}-\underline{A})^{-1}.\underline{b} \quad (14)$$

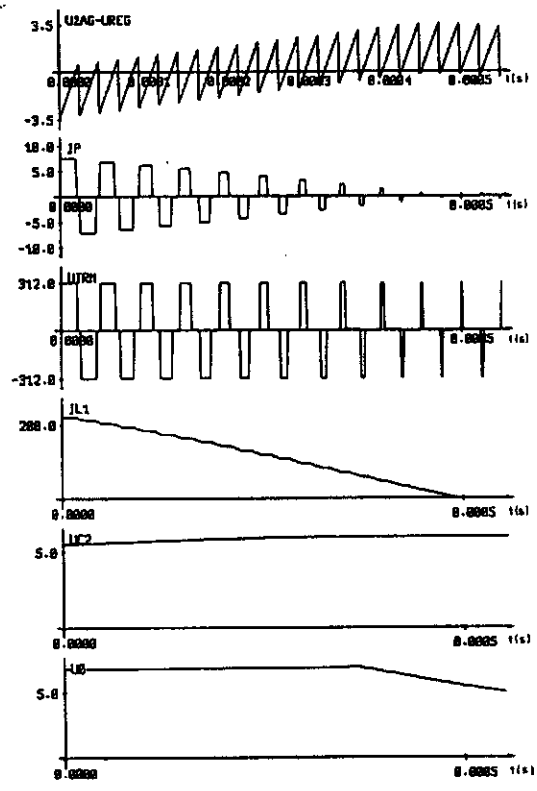
pri čemer velja za $\underline{A}$, $\underline{b}$ in $\underline{c}^T$ enačba (7), za $\underline{k}$ pa izraz

$$\underline{k} = (\underline{A}_1 - \underline{A}_2)\,\underline{X} + (\underline{b}_1 - \underline{b}_2)\,U_i \quad (15)$$

Sintezo regulatorja opravimo na podlagi znanih relacij linearne regulacijske tehnike [1, 2].

5. SIMULACIJA PULZNEGA USMERNIKA

Za ilustracijo poteka simulacije so prikazani časovni odzivi mostičnega tranzistorskega pulznega usmernika - slika 3. Za simulacijo smo uporabili programski paket PADSIM [6]. Sistem predstavimo z analogno blokovno shemo, ki jo v simbolični obliki vnesemo v program. Verzija programa PADSIM na računalniku VAX 11 nudi možnost sprotnega risanja časovnih potekov na ekranu terminala, časovne poteke pa si lahko narišemo tudi na koordinatnem risalniku. Takojšnja analiza rezultatov simulacije omogoča vrnitev v fazo podatkov, kjer lahko spremenimo konfiguracijo sistema (spremenimo vrednosti elementov) ali druge podatke (integracijski interval, celoten čas...). S spremenjenimi parametri simulacijo ponovimo. Postopek ponavljamo, dokler ne pridemo do želenih odzivov sistema oziroma optimalnih parametrov vezja.



Sl. 3: Časovni odzivi mostičnega napajalnika pri razbremenitvi (U_i = 312V, I_o = 200A → 20A, U_o = 5,5V)

5. LITERATURA

[1] R. D. Middlebrook, S. Čuk: Advances in switched mode powe conversion, Vol. 1 + 2, Teslaco, Pasadera, California 1981

[2] B. Alatič - Magistrska naloga, VTŠ Maribor, 1984

[3] J. Wüstehube: Schaltentzteile, Expert Verlag, Grafenau, 1979

[4] R. Adair: Ease switcher - transformer design using computer graphics, EDN, 13.10.1982, str. 147 - 149

[5] F. C. Y. Lee, Y. Yu: Computer - aided analysis and simulation of Switched DC - DC converters, IEEE Vol. IA - 15, No. 5, 1979, str. 511 - 520

[6] R. Svečko: Diplomska naloga, VTŠ Maribor, 1981

INTERAKTIVNI PROGRAMI ZA PRORAČUN NAMOTA U TRANSFORMATORU

Petar Čavlović,
"Rade Končar" - Elektrotehnički institut
Zagreb, Baštijanova bb.

UDK: 681.3.06

SAŽETAK: Prikazan je model interaktivnog programskog modula uz rad sa "čvrstim ekranom". Na temelju tog modela izrađeni su programi za proračun i slaganje namota u transformatoru: spiralnog, preloženog i EE - interleaved tipa. Programi su testirani na brojnim primjerima i dali su dobre rezultate, a primjena u praksi je u potpunosti opravdala ovakvu koncepciju programa za proračun i slaganje namota u transformatoru.

ABSTRACT: This work presents a modern concept of interactive program module with a "hard scren" feature. Based on this concept the programs for computer aided design of helical, disc, and EE-interleaved windings have been developed. This programs have been tested on many examples with excellent sucess. This utilization of these programs in praxis has verified that this concept is well suited for computer aided design of transformer windings.

1. UVOD

U članku je izložena koncepcija interaktivnih programskih modula. Interaktivni programski moduli temelje se na interaktivnom radu korisnika s programom (računalom) uz primjenu rada sa "čvrstim ekranom". Pod interaktivnim radom podrazumijeva se takav način rada čiju okosnicu čini dijalog korisnika s računalom. S druge strane rad uz "čvrsti ekran" podrazumijeva način rada kod kojeg korisnik ima subjektivni osjećaj da je slika na ekranu video terminala nepomična, a do njenog pomicanja ili obnavljanja dolazi tek na izričit zahtjev korisnika. Prema tim načelima razvijeni su programi za namote transformatora tipa: spiralni, preloženi, upleteni (EE-interleaved) i cilindrički. Postupak proračuna namota ide "korak po korak", naime projektant postupno slaže projektno rješenje namota. Zahvaljujući radu sa "čvrstim ekranom" ostvarena je mogućnost da u svakom trenutku projektant ima pred sobom dostignuto rješenje. Izradi programa prethodila je priprema podloga i algoritama za proračun namota te razvoj sustavske podrške za podržavanje interaktivnog rada uz nepomični ekran. Realizirani program za proračun i slaganje namota koriste se u svakodnevnom projektiranju u RO "Rade Končar - Transformatori" na širokom spektru transformatora. Dosadašnja iskustva su u potpunosti opravdala ovakav model korisničkih programa.

2. KONCEPCIJA INTERAKTIVNOG PROGRAMSKOG MODULA ZA SLAGANJE NAMOTA

Koncepcija interaktivnog programskog modula temelji se na interaktivnom radu korisnika s programom (računalom) uz primjenu rada sa "čvrstim ekranom". Pod interaktivnim radom podrazumijevamo takav rad čiju okosnisu čini dijalog korisnika s računalom u kojem se izmjenom pitanja i odgovora ostvaruje medusobna komunikacija. S druge strane rad sa "čvrstim ekranom" podrazumijeva način rada kod kojeg korisnik ima subjektivan osjećaj da je slika na ekranu video-terminala čvrsta, nepomična, a do njenog pomicanja ili obnavljanja dolazi tek na izričit zahtjev korisnika.

Ovakav model programa pokazao se za problem proračuna i slaganja namota kao najprihvatljiviji iz slijedećih razloga. Postupak slaganja namota ide "korak po korak", naime projektant postepeno slaže projektno rješenje namota. U tom postupku postoji niz mogućih povratnih veza, skretnica te je za takav problem interaktivni rad najprikladniji. Interaktivni rad se ostvaruje izmjenom pitanja i odgovora koja slijede u procesu slaganja namota i upravo ta jednostavna izmjena informacija odnosno medusobna komunikacija projektant-program (računalo) čini bitnu pretpostavku ovog programskog modela.

Ugrađujući u taj model rad sa "čvrstim ekranom" ovaj model programa dobio je kvalitet više. Naime radom sa "čvrstim ekranom" ostvaruje se mogućnost da u svakom trenutku projektant ima u vidu kompletno projektno rješenje, tj. ništa u procesu proračuna i slaganja namota mu ne "bježi" s vida. Upoznajmo se malo detaljnije s

koncepcijom interaktivnog programskog modula za slaganje namota.

Ulazni podaci potrebni za proračun namota upisuju se ili ručno preko terminala ili iz ulazne datoteke u "čvrstu" tablicu na ekranu.

Ulazni podaci potrebni za proračun namota su osnovni podaci transformatora odnosno namota. Niti jedan ulazni podatak nije izvedena veličina tj. veličina koju je potrebno prethodno posebno proračunati. To su podaci koje bi u pravilu projektant upisao listajući ugovor za taj transformator, odnosno standardne vrijednosti definirane propisom, standardom i dosadašnjom praksom i iskustvom u projektiranju, konstrukciji, tehnologiji i proizvodnji namota za velike transformatore. Neki od ulaznih podataka su "uvjetno" rečeno izvedeni podaci, naime proizlaze iz prethodnih proračuna tako da uvidom u izlazne liste tih proračuna nalazimo i njihove vrijednosti.

Korisnik upisuje ulazne podatke ručno preko terminala, odgovaranjem na postavljena pitanja koja se pojavljuju na dnu ekrana i tako postepeno formira "čvrstu" tablicu podataka na ekranu. Izgled "čvrste" tablice na ekranu sa svim oznakama prikazan je na slici 1. za programski modul za slaganje spiralnog namota.

Prilikom upisivanja ulaznih podataka u nekim slučajevima korisniku je ponuđen izbor dozvoljenih vrijednosti te ako upisana vrijednost nije iz tog izbora, pitanje se ponavlja dok nije upisana dozvoljena vrijednost. Taj izbor dozvoljenih vrijednosti je prvenstveno uvjetovan standardnim vrijednostima za neke veličine:

- frekvencija (50 ili 60 Hz),
- materijal namota (bakar ili aluminij),
- spoj namota (trokut ili zvijezda),
- broj faza (1 ili 3),
- spajanje namota (serijsko ili paralelno),
- tip namota (U-tip ili I-tip namota) za spiralni namot, itd.

Također u fazi upisivanja ulaznih podataka ako korisnik na traženu brojčanu vrijednost (npr. nazivna snaga, visina namota) upiše ništicu, pitanje se ponavlja.

Ovim raznim mehanizmima zaštite i uvjetovanošću upisa ulaznih podataka mogućnost pogreške kod upisa ulazniz podataka svedena je na minimum.

Po završetku upisa ulaznih podataka postoji mogućnost ispravka ili promjene bilo kojeg ulaznog podatka. Korisnik upisuje oznaku veličine (slika 1.) te se odmah na dnu ekrana pojavljuje odgovarajuće pitanje, kao kod upisa tog podatka, te sada korisnik upisuje vrijednost koju želi. Na taj način je pružena potpuna mogućnost kontrole upisa ulaznih podataka.

Nakon što je tako definitivno završen upis ulaznih podataka obrazuje se datoteka s tim ulaznim podacima prethodno odabranog (zadanog) imena. Ime datoteke može sadržavati najviše devet znakova, tipa:
SPR - za proračun spiralnog namota.

Datoteka ulaznih podataka obrazuje se sa ciljem da se izbjegne ponovno (ručno) upisivanje ulaznih podataka u slučaju kada korisnik ponavlja proračun namota s tim ili sličnim podacima, što je u procesu projektiranja normalna pojava, jer put do definitivnog projektnog rješenja traži od slučaja do slučaja višestruko ponavljanje cijelokupnog postupka.

Obzirom da je struktura ulaznih podataka takva da je ručno upisivanje ulaznih podataka potrebno samo jedanput, u pravilu kod prvog slaganja namota, a u daljnjem procesu projektiranja namota potrebne su vrlo male promjene ulaznih podataka (npr. visina namota, unutarnji promjer namota itd.), ponavljanje upisivanja bio bi gubitak vremena.

Dakle pri pozivanju programa za slaganje namota, te ako se želi upis ulaznih podataka iz ulazne datoteke, automatski se podaci potrebni za proračun pohranjeni u traženoj datoteci upisuju u "čvrstu" tablicu na ekran terminala. Ukoliko tražena datoteka ne postoji korisnik mora ulazne podatke upisati ručno.

NAM =		NF =	MAT =	GSP =
SN =	UN =	F =	NST =	SI =
U =	S =	V(S)=	DU =	HN =
DELI=	ARK =	EPS =	NL =	DELS=
TIP =	ULAZ=	HLAD=	SPOJ=	
IF =	GAMA=	Q =	H =	B =
P =	PA =	PR =	ARK =	NRKU=
APR =	NPR =	KD =	AAK =	NAK =
PUK =	ZAGR=		AN =	DV =
HN =	HNM =			

Slika 1. Izgled čvrste tablice na ekranu sa svim oznakama za proračun spiralnog namota

Završavajući upis ulaznih podataka prelazi se na proračun i slaganje namota. Slaganje namota je interaktivni rad korisnika s programom. Tijekom proračuna namota pojavljuje se na ekranu niz izračunatih parametara i pitanja na koja je korisnik dužan odgovoriti. Projektant prihvaća ili odbija ponuđenu izračunatu vrijednost, upisuje odabranu vrijednost iz niza ponuđenih vrijednosti (parametri vodiča, širina izolacije), odabire moguć put u slaganju namota te na taj način omogućuje odvijanje programa. Zahvaljujući lakoj i razumljivoj komunikaciji projektant-program, detaljnoj podlozi i algoritmima za proračun namota vrlo brzo se ostvaruje kvalitetno projektno rješenje.

Detaljno opisivanje tijeka proračuna i slaganja pojedinog namota, odvelo bi nas vrlo daleko, te bi zapravo demonstracija "uživo" dala pravu sliku interaktivnih programskih modula za slaganje namota.

Dio izračunatih parametara u procesu slaganja namota upisuje se u nastavku "čvrste" tablice na ekranu, te je na taj način ostvarena mogućnost da projektant u svakom trenutku ima uvid u kompletno projektno rješenje. Definitivan izgled "čvrste" tablice na ekranu za primjer slaganja spiralnog namota prikazan je na slici 2.

Nakon što je završeno slaganje namota postoji mogućnost da se cijeli postupak ponovi s novim ulaznim podacima ili projektant može s istim ulaznim podacima slagati neko drugo projektno rješenje namota.

Izlazni podaci proračuna pohranjuju se u izlaznu datoteku tipa TMP (datoteka privremenog karaktera), istog imena kao i ulazna datoteka u konkretnom slučaju, i služe za dokumentiranje projektnog rješenja.

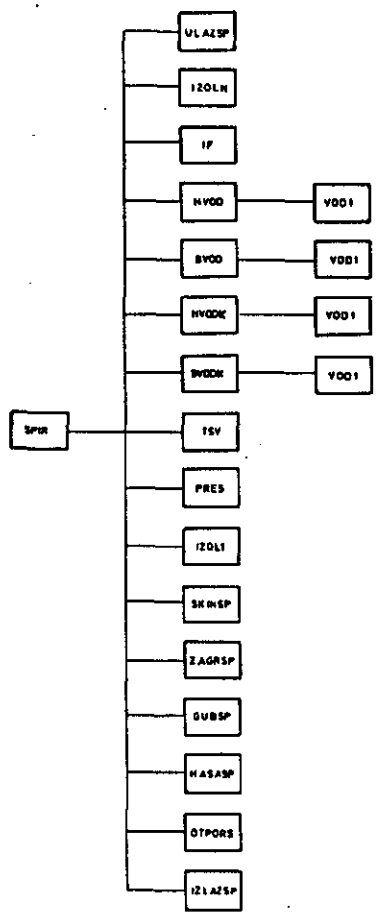
U programe za proračun namota ugrađen je veći broj standarda i propisa koji čine tzv. popratnu papirnatu dokumentaciju koja je potrebna prilikom projektiranja namota i time je rad projektanta bitno olakšan. Umjesto da pretražuje po papirima prilikom proračuna, njemu se u procesu slaganja nudi izbor dozvoljenih (standardnih) vrijednosti.

```
NAM = NN                          NF  =3          MAT =CU         GSP =D
-------------------------------------------------------------------------------
SN  =  340.00  UN  =   11.00  F   =   50.00  NST =    3.00  SI  =   24.00
W   =   48.00  S   =    2.00  V(S)=    1.00  DU  = 1153.00  HN  = 2590.00
DELI=    0.52  ARK =    4.20  EPS =    1.00  NL  =   40.00  DELS=   20.00
-------------------------------------------------------------------------------
TIP =U         ULAZ=1         HLAD=OFAF
===============================================================================
IF  =10303.03  GAMA=    3.42  Q   =  501.77  H   =   13.14  B   =   45.15
P   =       6  PA  =       6  PR  =       1  ARK =    4.20  NRKU=       5
                              KD  =    1.08
PUK =  462.00  ZAGR=   21.10                  AN  =  111.50  DV  = 1376.00
HN  = 2590.60  HNM = 2651.45

>
```

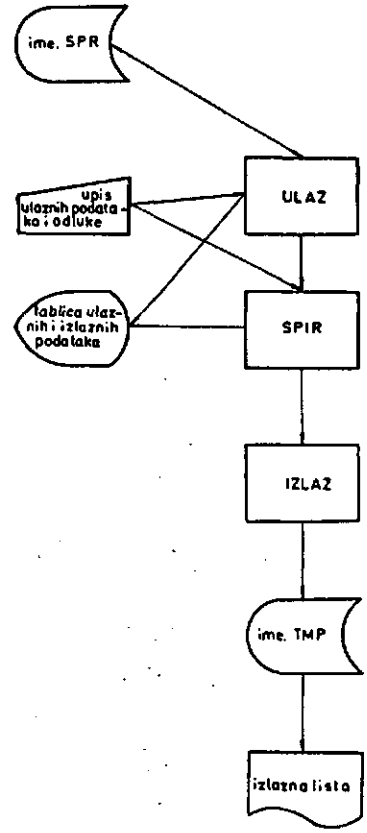
Slika 2. "Čvrsta" tablica na ekranu za primjer proračuna spiralnog namota

3. OPIS PROGRAMA ZA PRORAČUN SPIRALNOG NAMOTA

Programi za proračun namota tipa spiralni, preloženi i upleteni realizirani su na istim načelima. Ovdje je prikazana hijerarhijska karta koja pokazuje logišku povezanost modula i tijek podataka u programu za proračun spiralnog namota. Dan je također i kratki opis programskih modula.



Slika 3. Hijerarhijska karta za program za slaganje spiralnog namota



Slika 4. Tijek podataka u programu za slaganje spiralnog namota

OPIS PROGRAMA KOJI SE KORISTE U PROGRAMU ZA SLAGANJE SPIRALNOG NAMOTA

SPIR - Glavni program.

ULAZSP - Potprogram za:
- ispis ulaznih podataka u "čvrstu" tablicu na ekranu ili
- upis ulaznih podataka u "čvrstu" tablicu na ekranu i
- ispis ulaznih podataka u ulaznu datoteku

IZOLN - Potprogram za odredivanje nominalnog prirasta izolacije vodiča

IF - Potprogram za proračun fazne struje

HVOD - Potprogram kojim se određuje u okolišu izračunate visine vodiča standardna vrijednost visine vodiča. Ovaj potprogram koristi potprogram VOD1

BVOD - Potprogram kojim se određuje u okolišu izračunate širine vodiča standardna vrijednost širine vodiča. Ovaj potprogram koristi potprogram VOD1.

HVODK - Potprogram kojim se određuje u okolišu izračunate visine vodiča standardna vrijednost visine vodiča, ako je vodič motan na "kant". Ovaj potprogram koristi potprogram VOD1.

BVODK - Potprogram kojim se određuje u okolišu izračunate širine vodiča standardna vrijednost širine vodiča, ako je vodič motan na "kant". Ovaj potprogram koristi potprogram VOD1.

PRES - Potprogram za proračun presjeka vodiča (za profilni vodič).

TSV - Potprogram za odredivanje standardnih vrijednosti transponiranog vodiča:
- visina temeljnog vodiča,
- visina vodiča,
- širina temeljnog vodiča,
- širina vodiča,
- broj temeljnih vodiča
- presjek vodiča.

IZOL1 - Potprogram kojim se određuje prirast izolacije vodiča (radijalno i aksijalno).

SKINSP - Potprogram za proračun faktora skin-efekta.

ZAGRSP - Potprogram za proračun srednjeg zagrijanja namot-ulje.

GUBSP - Potprogram za proračun gubitaka u namotu (I^2R i P_{Cu}).

MASASP - Potprogram za proračun mase namota (goli i izolirani vodič).

OTPORS - Potprogram za proračun otpora namota (po stupu).

VOD1 - Potprogram kojim se određuje polje standardnih vrijednosti visine ili širine vodiča.

IZLAZSP - Potprogram za ispis izlaznih podataka u izlaznu datoteku.

4. ZAKLJUČAK

Koncepcija interaktivnog programa uz rad sa "čvrstim ekranom", koja je primjenjena u izgradnji programa za slaganje i proračun namota, je u potpunosti opravdala idejne odrednice. Jednostavnost, lakoća, brzina i komfor u procesu slaganja namota, rad sa "čvrstim ekranom" pokazala su se kao bitna svojstva na kojima treba temeljiti i daljnji rad na razvoju programa za proračun preostalih tipova namota.

Ugradnja propisa, standarda te oslobađanje projektanta od raznovrsne dokumentacije u procesu slaganja namota pokazuje se kao daljnji kvalitet programa. Također bi upravo na ovoj koncepciji trebala počivati realizacija programa koji su karakterom problema srodni problemu slaganja namota, dakle takav vid problema u kojima bi koncept interaktivnog programa potvrdio svoje vrijednosti.

LITERATURA

Petar Čavlović: Razvoj interaktivnih programskih modula za proračun i slaganje namota u transformatoru, Magistarski rad, Elektrotehnički fakultet Sveučilišta u Zagrebu, 1983. g.

PROGRAMSKI SUSTAV ZA PRORAČUN ELEKTRIČNOG POLJA UZ NAMOTE TRANSFORMATORA

Lozica Mladen,
Elektrotehnički institut "Rade Končar"
Zagreb, Baštijanova b.b.

UDK: 681.3.06

U članku je opisan programski sustav za proračun električnog polja u prvom aksijalnom kanalu uz visokonaponski namot transformatora za slučaj ispitivanja udarnim naponom. Kao osnova sustava korišten je program za proračun električnog polja u osnosimetričnim modelima baziran na primjeni integralnih jednadžbi.

PROGRAM SYSTEM FOR CALCULATION OF THE ELECTRICAL FIELD NEAR TRANSFORMER WINDINGS - This paper presents the program system for calculation of the electrical field in the first axial chanell near the high - voltage transformer winding, in the case of the surge voltage test. The base of the system is the program which utilises integral-equations approach for calculation of electrical fields in axially-symmetric models.

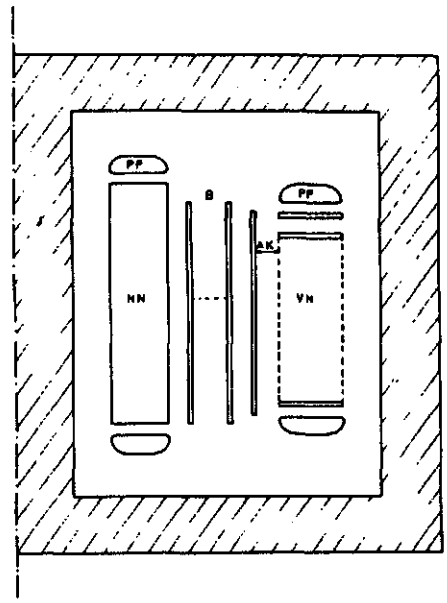
1. UVOD

Postojeći program za proračun električnih polja razvijen je u Sektoru za transformatore Elektrotehničkog instituta "Rade Končar" u suradnji s Elektrotehničkim fakultetom u Zagrebu. Program je zasnovan na rješavanju osnosimetričnih modela primjenom integralnih jednadžbi. Općenitost programa omogućuje rješavanje bilo kakve osnosimetrične strukture. To, međutim , uvjetuje i uopćen način zadavanja ulaznih podataka koji predpostavlja poznavanje funkcioniranja metode proračuna. Uz to bilo je potrebno da korisnik donekle poznaje i operacioni sistem samog računala.

Ukoliko se proračun ograniči na model čija se geometrija bitno ne mijenja, moguće je pojednostaviti zadavanje ulaznih podataka. U daljnjem tekstu biti će opisano definiranje modela i ulano-izlazni moduli koji omogućavaju znatno jednostavnije korištenje programa prilikom projektiranja.

2. DEFINIRANJE MODELA

Na slici 2.1 prikazana je geometrija od koje se polazi prilikom definiranja modela.



Sl. 1 - skica presjeka geometrije na osnovu koje se formira model

Značenje oznaka je slijedeće:

J - jezgra transformatora

NN - niskonaponski namot
VN - visokonaponski namot prikazan razdjeljen na pojedine svitke
PP - potencijalni prstenovi na krajevima namota
B - papirnate barijere u međunamotnom prostoru
AK - prvi aksijalni kanal uz VN namot.

Kada bi memorija računala dopuštala, geometrija na slici 1 mogla bi se nepromjenjena koristiti kao model za proračun el. polja. Proračun, upravo zbog ograničene memorije, dozvoljava rješavanje relativno jednostavnih modela, pa se geometrija na slici 1 mora pojednostaviti. U ovom slučaju traži se proračun polja samo u prvom aksijalnom kanalu uz VN namot pa se model pojednostavljuje na temelju slijedećih kriterija:

- polje u prvom aksijalnom kanalu računa se u visini prvih šest ulaznih svitaka VN namota
- sve papirnate barijere u međunamotnom prostoru, osim prve uz VN namot, spajaju se u rezultantnu barijeru
- sve konture na jednakom potencijalu spajaju se zajedno u jednu konturu.

Da bi model bio potpuno definiran korisnik unosi niz parametara koji određuju tip svitaka u VN namotu, mjesto izvoda faze iz VN namota, te niz podataka o dimenzijama transformatora.

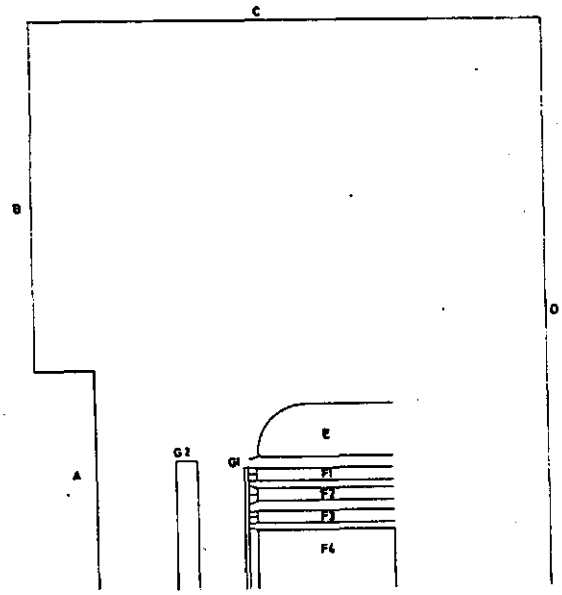
3. ULAZNO - IZLAZNI MODULI

Ulazno-izlazni moduli su napravljeni tako da maksimalno olakšaju korištenje programa, odnosno rad s računalom. Na temelju kriterija spomenutih u prethodnom odjeljku ulazni moduli formiraju model, diskretiziraju ga i pripremaju podatke za daljnji proračun polja i za izlazne module. Program ujedno proračunava raspodjelu potencijala na svicima VN namota za poznate vrijednosti udarnog vala. Pretpostavka je da se VN namot aproksimira samo s mrežom kapaciteta, a za udarni val se uzima odrezani oblik. Po želji korisnika potencijali se mogu zadavati i posebno. Ulazni moduli vrše i kontrolu ulaznih podataka, tako da je spriječeno unošenje nelogičnih podataka. Broj ulaznih podataka bitno je smanjen u odnosu na broj koji je bio potreban prilikom rada s izvornim programom.

Izlazni moduli prilagođeni su za prikaz rezultata proračuna za specifični slučaj računa polja u prvom aksijalnom kanalu. Za projektanta je bitno dobiti faktore sigurnosti (definirani kao omjer dozvoljenog i srednjeg polja po silnici) za tipične silnice u prvom aksijalnom kanalu koje kreću s prva tri ulazna svitka. Na osnovu toga razrađeni su izlazni moduli koji omogućavaju:

- Prikaz modela i silnica na ploteru
- Tablični prikaz faktora sigurnosti na listi štampača, ili na video-terminalu.

Za primjer proračuna uzet je transformator ARZ 150 000/420, te je na slici 2 prikazana geometrija modela iscrtana na ploteru i silnice ucrtane u prvi aksijalni kanal. Radi se o transformatoru s VN namotom tipa EE interlieved i izvodom faze s gornje, vanjske strane VN namota.



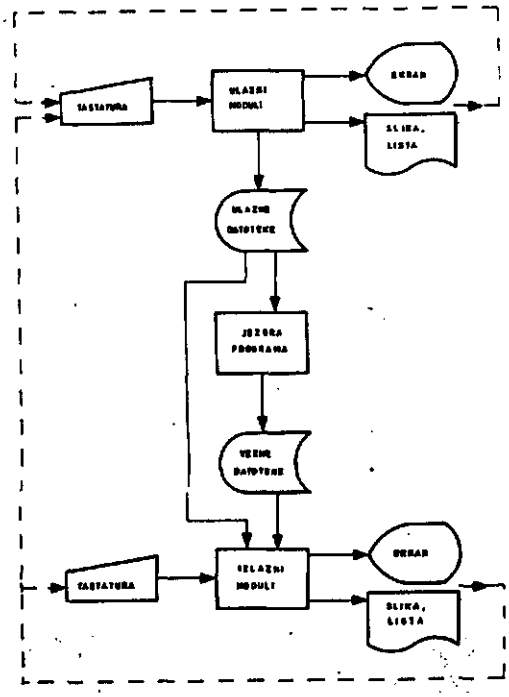
Sl. 2 - slika modela dobivena na ploteru s ucrtanim silnicama

Značenje oznaka je slijedeće:

A - niskonaponski namot na 0% potencijala
B - stup jezgre na 0% potencijala
C - jaram na 0% potencijala
D - kotao na 0% potencijala
E - potencijalni prsten na 100% potencijala
F1,...,F3 - prva tri ulazna svitka na 97,23; 98,99 i 93,59% potencijala
F4 - daljnja tri svitka spojena u jednu konturu s linearnom podjelom napona 95,26 -86,73% potencijala od vrha konture do dna
G1 - prva papirnata barijera do VN namota
G2 - rezultantna papirnata barijera smještena u sredinu međunamotnog prostora.

4. ORGANIZACIJA SUSTAVA

Na slici 3 prikazana je organizacija sustava i tok procesa proračuna. Program je zamišljen da radi interaktivno, pa je korisniku omogućena jednostavna izmjena ulaznih podataka i kontrola toka procesa. Nakon izvođenja ulaznih modula i kontrole formiranog modela izvodi se jezgra programa i pomoću izlaznih modula dobiva se traženi oblik prezentacije izlaznih podataka. Kompletno vođenje toka procesa povjereno je procesoru stvorenom u tu svrhu, tako da se korisniku što je više moguće olakša rad s računalom. Na slici 3 crtkanim linijama su označeni mogući izbori u pojedinim fazama izvođenja programa. Sustav je rađen modularno tako se jednostavno može servisirati i proširivati novim modulima.



Sl. 3 - organizacija sustava

5. ZAKLJUČAK

Opisani sustav za proračun polja u međunamotnom prostoru transformatora omogućava korisniku jednostavno zadavanje ulaznih podataka, te obradu više različitih varijanti geometrije, uz minimalnu izmjenu ulaznih podataka. Nije potrebno da korisnik detaljnije poznaje proračunni operacioni sistem računala. Izlazni podaci prilagođeni su zahtjevima projektanta i mogu se direktno dalje koristiti. Modularna organizacija sustava omogućava jednostavno mijenjanje starih, ili dodavanje novih modula.

6. LITERATURA

L1 - Haznadar Z., Đurek M.: Numeričko rješavanje elektrostatskih osnosimetričnih polja, IV Bosanskohercegovački simpozijum iz informatike, Jahorina 1980.

L2 - Štih Ž.: Programski sustav za proračun električnih polja u transformatorima, 4. znanstveni skup PPPR, Stubičke Toplice, 1982.

L3 - Štih Ž.: Proračun električnih polja u transformatorima visokog napona s pomoću integralnih jednadžbi i polinoma trećeg stupnja, magistarski rad, Zagreb 1981.

L4 - Štih Ž., Lozica M., Papić N.: CAD sustav za projektiranje izolacije velikih transformatora, 6. znanstveni skup PPPR, Zagreb, 1984.

GRAFIKA IN CAD/CAM SISTEMI

GRAPHICS AND CAD/CAM SYSTEMS

SOFTVER GRAFIČKOG TERMINALA PREMA GKS STANDARDU

Vraneš Predrag, Miloš Bajčetić
Institut Mihajlo Pupin, Beograd, Yugoslavia

UDK: 681.3.06

Realizovan je grafički softver za terminal srednje rezolucije. Softver je projektovan kao paket procedura kojima se obezbedjuje veza izmedju aplikacionih programa i hardvera terminala. Deo programa se izvršava na host računaru dok se ostatak izvršava pod kontrolom mikroprocesora u terminalu. Sistem je zasnovan na GKS standardu i podržava nivo 0a.

A GKS BASED GRAPHICS TERMINAL SOFTWARE

A graphics software for the medium resolution terminal has been implemented. It is designed as a procedure package that provides an interface between an application program and graphics hardware. One part of the package runs on the host computer while the rest of it runs under the control of a microprocessor located in the terminal. The system is based on the GKS standard conforming level 0a.

1. UVOD

U radu će biti prikazana realizacija grafičkog softvera za terminal TIM-001. Ovaj terminal je VT-100 kompatibilan uredjaj zasnovan na Intel-ovoj tehnologiji. Grafički modul, koji je opciono proširenje terminala obezbedjuje bit-mapiranu grafiku rezolucije 544 x 240 piksela sa mogućnošću prikazivanja 4 nijanse sivog. Softverski paket za ovaj modul se sastoji od niza standardnih procedura kojima se obezbedjuje sprega izmedju aplikativnih programa i hardverskih specifičnosti terminala. Jedan od osnovnih zadataka softvera je da se programeru stavi na raspolaganje takav alat koji obezbedjuje protabilnost aplikacija i nezavisnost nihove izrade od hardverske organizacije grafičkog terminala.

U tu svrhu, sve implementirane funkcije su realizovane prema GKS standardu. GKS (Graphical Kernel System) je zvanično, u poslednje vreme široko prihvaćen internacionalni standard,a sastoji se od niza funkcija kojima se omogućuje jednostavno programiranje grafičkih aplikacija.

Trenutno realizovanim funkcijama,ovog projekta,nije obuhvaćen rad sa ulaznim uredjajima kao ni koncept segmenata što prema klasifikaciji, datoj GKS standardom,predstavlja nivo 0a. Korisnički program pisan u nekom od viših jezika povezuje se sa grafičkim paketom standardnim pozivanjem procedura. GKS-om je potpuno sintaktički obuhvaćeno povezivanje sa programima pisanim U PASKAL-u i FORTRAN-u.

Radi opterećenosti procesora u samom terminalu paket je realizovan jednim delom na "host" računara dok je preostali deo implementiran u firmveru samog terminala.

2. OPŠTE KARAKTERISTIKE HARDVERSKO-SOFTVERSKE ORGANIZACIJE GRAFIČKOG TERMINALA

U osnovnoj konfiguraciji TIM-001 predstavlja alfanumerički VT-100 kompatibilan terminal sa mogućnošću prikazivanja 80 karaktera u 24 reda. Hardverska koncepcija terminala bazirana je na Intel-ovoj tehnologiji, pre svega mikroprocesoru 8085 i CRT kontroleru 8275. Iako u jednom od režima rada omogućuje generisanje osnovnih grafičkih primitiva komponovanjem grafičkih karaktera (semi grafika) sistem je u cilju poboljšanja grafičkih mogućnosti proširen dodatnim modulom. Pod poboljšanjem misli se pre svega na mogućnost generisanja kompleksnijih scena kao i na jednostavniju izradu aplikacionih programa. Pored toga, od softvera se očekivalo da korisniku učini hardver transparentnim kao i da obezbedi portabilnost aplikacionog programa,odnosno mogućnost njegovog izvršavanja na drugim grafičkim sistemima.

Dodatni modul omogućuje rad terminala u bit-mapiranom grafičkom režimu rada sa rezolucijom od 544 x 240 piksela i dva bita informacije po pikselu. Glavna komponenta grafičkog modula je grafički kontroler INTEL 82720 koji pod kontrolom mikroprocesora generiše osnovne sinhronizacione signale i signale za upravljanje prikaznom (displej) memorijom. Detaljnije informacije o hardverskoj organizaciji modula mogu se naći u posebnom radu [1]. Grafički kontroler takodje, raspolaže setom naredbi za generisanje osnovnih grafičkih primitiva, što pojednostavljuje izradu softvera (firmvera) i rasterećuje procesor. Kontroler može da crta tačku, vektor, luk, pravougaonik, ispisuje karaktere i postavlja atribute kojima se definiše izgled ovih primitiva. Mikroprocesor preko 8-bitne magistrale šalje kontroleru naredbe sa odgovarajućim parametrima čijom intepretacijom kontroler vrši iscrtavanje primitiva bez dodatnog angažovanja procesora. Njihov dijalog se obavlja preko 16-bajtnog FIFO bafera kontrolera.Pre slanja podataka potrebno je proveriti status bafera jer u slučaju da je pun može doći do gubitka prethodno poslatih informacija.

Relativno skromna procesorska moć mikroprocesora 8085 uz stalno angažovanje na poslovima vezanim za alfanumerički režim rada i komunikaciju sa hostom ostavlja malo prostora za realizaciju grafičkih funkcija. To je osnovni razlog što je softver implementiran u dva nivoa, deo koji se izvršava na host računaru i jezgro implementirano u firmveru terminala.

Grafičke funkcije kojima raspolaže korisnik su deo standardnih GKS funkcija, [2]. Izabran je ovaj standard jer u potpunosti odgovara nameni i projektnim zahtevima, a uz to je u poslednje vreme široko primenjivan kao zvanično prihvaćeni internacionalni standard. Prednost njegove upotrebe u odnosu na nestandardno realizovan grafički softver se može uporediti sa prednošću korišćenja viših programskih jezika u odnosu na asembler. Aplikativni programi pisani po ovom standardu su prenosivi sa mašine na mašinu čime se po cenu optimalnog koda ostvaruje jedan od osnovnih ciljeva savremeno koncipiranih grafičkih računarskih sistema, povećanje produktivnosti programerskog rada.

Implementacijom dela softverskog paketa u firmveru samog terminala i izborom GKS standardnih funkcija u datim okolnostima pokušalo se udovoljiti i ostalim zahtevima

savremeno koncipiranih grafičkih terminala [3], rasterećenje hosta od grafičkih poslova i portabilnost aplikativnog softvera.

3. OSNOVNE FUNKCIJE IMPLEMENTIRANOG GKS STANDARDA

GKS se može definisati kao biblioteka grafičkih programa kojima se omogućuje generisanje dvodimenzionalnih (2D) slika nezavisno od sistema na kome se izvršavaju. Tako se, kako je već istaknuto, ostvaruje funkcionalna sprega izmedju aplikacionog programa i u našem slučaju crno-belog monitora kao izlazne jedinice.

Kompletan set grafičkih funkcija GKS-a obuhvata i rad sa grafičkim sistemima visokog kvaliteta. Kod sistema čija je organizacija skromnija, kao u ovom slučaju, dovoljno je koristiti samo podskup iz kompletnog seta funkcija standarda. To je GKS-om i predvidjeno tzv. konceptom struktura različitog nivoa. Definisano je 12 nivoa koji odgovaraju različitim zahtevima grafičkih aplikacija. Nivoi su kompatibilni naviše. Struktura nivoa bi se mogla predstaviti sa 2 nezavisne ose, od kojih bi jedna predstavljala ulazne nivoe (u opsegu od a do c) a druga izlazne (m, 0, 1, 2). Naša implementacija podržava funkcije nivoa 0a, [2].

Nivo 0a obuhvata funkcije izlaza svih grafičkim primitiva, upravljanje njihovim izgledom posredstvom atributa i upravljanje transformacijama koordinatnih sistema. Grafički ulaz i rad sa segmentima nisu podržani.

Od izlaznih primitiva korisniku su na raspolaganju:

- POLYLINE - niz nadovezanih linija
- POLYMARKER - niz simbola na odredjenim pozicijama
- FILL AREA - poligonalna površina
- TEXT - niz karaktera

Izgled navedenih primitiva zavisi od vrednosti atributa. Posebnom funkcijom korisnik upisuje vrednost svakog od atributa u listi stanja. U toku procesa crtanja atributi se uzimaju iz 2 izvora. To su, već pomenuta lista stanja i unapred definisane tablice (bundle tables). Koji je izvor aktuelan odredjuju indikatori izbora atributa.

Atributi primitiva su:

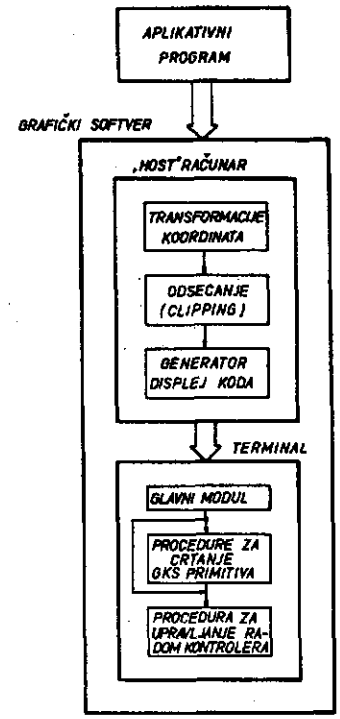
- POLYLINE - indeks, tip, boja i debljina
- POLYMARKER - indeks, tip, boja i veličina
- FILL AREA - indeks, način ispunjavanja i boja
- TEXT - indeks, preciznost, veličina, razmak, boja, vektor položaja, položaj pravougaonika koji uokviruje tekst.

Kada se atributi odredjuju vezano, indeks svake primitive ukazuje na vrstu u tablici iz koje se očitavaju vrednosti atributa, a kada ih odredjujemo pojedinačno očitavaju se iz liste stanja. Na ovom nivou korisnik ne može definisati nove vrste u tablicama.

4. REALIZACIJA SOFTVERA

Programski paket, koji se izvršava na "host" računaru omogućuje spregu izmedju aplikativnog programa i terminala. Taj paket se, kao biblioteka potprograma, povezuje sa aplikativnim programom pisanim u nekom od viših programskih jezika. Standardom je potpuno definisano povezivanje sa FORTRAN i PASCAL programima. Korisnik u svom programu poziva pojedine potprograme, a na "host" računaru se vrši interpretacija poziva.

Blok dijagram grafičkog softvera prikazan je na sl. 1. Od svih funkcija predvidjenih pomenutim nivoom GKS-a, na "host" računaru se obavlja transformacija koordinata okoline (world coordinates) u normalizovane koordinate i odsecanje (clipping) u odnosu na specificirano vidno polje. Posebnom funkcijom korisnik definiše svoj koordinatni prostor i deo normalizovanog koordinatnog prostora na koji se njegov prostor preslikava. Normalizovane koordinate su realni brojevi od 0 do 1. Odsecanje se može onemogućiti upisom odgovarajuće vrednosti u registar odsecanja. Po obavljanju transformacija koordinata i odsecanju nevidljivih delova slike formira se displej fajl koji se uglavnom sastoji od funkcija za generisanje izlaznih primitiva sa normalizovanim parametrima kao koordinatama. Zahtevom za interpretaciju od strane hosta vrši se izvršavanje funkcija iz displej fajla.



Slika 1. Blok-dijagram grafičkog softvera

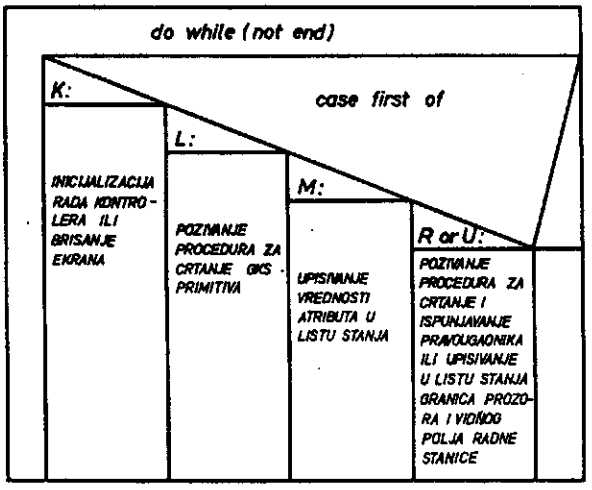
Kako su te funkcije u tesnoj vezi sa hardverom koji generiše sliku njihovo izvršavanje se obavlja u samom terminalu. Kao rezultat interpretacije na hostu terminalu se preko serijske magistrale šalje jednoznačni kôd funkcije sa odgovarajućim parametrima.

Parametri mogu biti realni brojevi (normalizovane koordinate) i celi brojevi (vrednosti atributa primitiva u ASCII kodovima karaktera). U meri u kojoj je to bilo moguće u kodiranju je poštovan Tektronix 4110 standard. Kôd svake funkcije se sastoji od 'escape' simbola i još 2 ASCII kodirana karaktera. Simbol 'escape' označava i početak tekuće naredbe i kraj prethodne. Za one funkcije GKS koje Tektronix ne podržava usvajani su nepostojeći kodovi.

Deo softvera realizovanog u terminalu je smešten u ERPOM-u i izvršava se pod kontrolom lokalnog mikroprocesora 8085. Ovaj deo grafičke biblioteke je izvorno pisan na višem programskom jeziku za Intel-ove mikroprocesore PL/M-80 [4]. Taj jezik ima dobar repertoar kontrolnih struktura i podržava koncept modularnosti.

Firmver se može razložiti u 3 nivoa: glavni modul, procedure za realizaciju GKS primitiva i procedure za upravljanje kontrolerom. U glavnom modulu se vrši obrada podataka koji stižu preko serijske magistrale sa host računara. Iz glavnog modula pozivaju sve procedure za crtanje GKS primitiva i neke od procedure za upravljanje kontrolerom.

Glavni modul (slika 2) se sastoji od beskonačne petlje u čijem jednom izvršavanju se analizira sekvenca koda i parametara pristiglih izmedju 2 'escape' simbola. U ovoj petlji nalazi se "do-case" struktura u kojoj se na osnovu pristiglih kodova odlučuje kakvu akciju preduzeti. Mogu se pozivati procedure ili obavljati upis vrednosti atributa u listu stanja. Lista stanja sadrži vrednosti svih atributa, unapred definisane tablice primitiva i granice prozora i vidnog polja radne stanice. Od procedura za upravljanje kontrolerom pozivaju se procedure



Slika 2. Strukturni dijagram glavnog modula

za inicijalizaciju rada kontrolera i brisanje ekrana i one nemaju parametre. Blokovi u kojima se pozivaju procedure za crtanje GKS primitiva i upisivanje vrednosti atributa u listu stanja su takodje "do case" blokovi ugnježdeni u prethodnom.

Ako su parametri koji dolaze normalizovane koordinate, na njima se obavlja transformacija radne stanice. Korisnik definiše prozor radne stanice kao deo normalizovanog koordinatnog prostora kojeg želi da prikaže i vidno polje radne stanice koje predstavlja deo ekrana na kome se normalizovani prostor prikazuje. Transformacija radne stanice preslikava normalizovane koordinate u koordinate ekrana (piksela).

Procedure za crtanje primitiva u toku svog izvršavanja pozivaju procedure nižeg nivoa koje upravljaju radom kontrolera. Procedure za upravljanje kontrolerom se uglavnom sastoje od operacionih kodova naredbi kontrolera i pridruženih parametara koji se upisuju u njegov FIFO - - bafer. Te procedure postavljaju grafički kursor na željenu poziciju, upisuju uzorak linije ili karaktera u parametarski RAM kontrolera, ispisuju karakter, crtaju linije i tačke. Procedure koje crtaju primitive u početku izvršavanja ispituju indikatore izvora atributa i na osnovu njihovog sadržaja odredjuju vrednosti. Posle ovoga se ispituje registar sadržaja parametarskog RAM-a i poziva procedura za upisivanje željenog uzorka u parametarski RAM. Vodjenjem evidencije o sadržaju ovog RAM-a izbegava se ponovan upis aktuelnog uzorka. Dalje se crtanje obavlja pozivanjem procedure za crtanje jednostavnijih geometrijskih oblika (tačke, linije).

Primitive za crtanje i ispunjavanje pravougaonika spadaju u one funkcije GKS-a koje nisu zahtevane standardom i čiji je način realizacije samo konceptualno dat. Njihova realizacija je iskoristila već prisutne mogućnosti kontrolera.

5. ZAKLJUČAK

U radu je prikazana realizacija grafičkog softvera za terminal TIM-001 projektovan u Institutu Mihajlo Pupin. U cilju obezbedjenja portabilnosti pisanih aplikacija, lakšeg programiranja i nezavisnosti aplikacije od hardverske organizacije korišćen je GKS standard nivoa 0a. Zahtev savremeno koncipiranih grafičkih sistema za rasterećenjem hosta od poslova vezanih za grafiku je poštovan u meri u kojoj to dozvoljava procesorska moć mikroprocesora u terminalu (Intel 8085). Sistem je namenjen 2D tehničkim aplikacijama nižeg nivoa ili složenijim poslovnim aplikacijama. Obzirom da je deo softvera realizovan u terminalu zaokružen a i da se ne može dalje proširivati zbog opterećenja mikroprocesora 8085 sva eventualna proširenja bi trebalo implementirati na host računaru. Moguće je uvesti rad sa segmentima, dozvoliti korisniku definisanje vrsta u tablicama atributa i obezbediti podršku ulaznih uredjaja (svetleće pero ili tablet). Realizacijom ovih funkcija bi se ostvario viši nivo GKS standarda. Kako je kompletan softver realizovan u jeziku PL/M-80 moguće ga je u znatnoj meri upotrebiti kod daleko kompleksnijih sistema na čemu se u Institutu Mihajlo Pupin i radi.

7. REFERENCE

1. Predrag Vraneš, Ivan Vojvodić, "Projekt terminalskog grafičkog modula srednje rezolucije", INFORMATICA ' INFORMATICA '85.

2. "Graphical Kernel System", Computer Graphics (ACM/Siggraph), Vol.18, No.2, Feb. 1984.

3. James Warner, "Standard Graphics Software for High-Performance Application", IEE CG & A, March 1985.

4. PL/M-80, Programming Manual, INTEL.

PROJEKAT TERMINALSKOG GRAFIČKOG MODULA SREDNJE REZOLUCIJE

Vraneš Predrag, Ivan Vojvodić
Institut Mihajlo Pupin, Belgrade, Yugoslavia

UDK: 681.3.014

U radu je prikazan projekt rasterskog grafičkog modula srednje rezolucije. Modul je predvidjen za ugradnju u CRT terminal TIM-001, ali može se upotrebiti sa bilo kojim mikroprocesorom kontrolisanim uredjajem. Sistem se bazira na VLSI grafičkom displej kontroleru INTEL 82720.Omogućen je izbor jednog od dva režima rada - crno beli režim ili režim rada u boji (4 boje,odnosno 4 nivoa sivog). Modul podržava GKS grafički standard nivoa OA.

DESIGN OF A MIDDLE RANGE RESOLUTION GRAPHICS TERMINAL MODULE

The design of a raster-scan graphics module with a middle range resolution is shown. This module is intended for use with a CRT terminal TIM-001, but it can be used with any microprocessor driven system. The system is based on a VLSI graphics display controller INTEL 82720. It offers two modes of operation - black and white or color-mode (4 colors or 4 gray scale levels). GKS standard level OA is supported.

1. UVOD

Računarska grafika, kao jedno od najneposrednijih sredstava komuniciranja čoveka sa računarom postaje ne samo neizbežni deo većine računarskih sistema, već i značajan faktor pri oceni njihovog kvaliteta.

Pojavom VLSI integrisanih kola sa grafičkim mogućnostima - grafičkim kontrolerima, omogućen je razvoj relativno jeftinih rasterskih grafičkih terminala srednje i visoke rezolucije za primenu u CAD/CAM sistemima.

Rukovodjeni zahtevima tržišta kao i mogućnošću brze realizacije, u Institutu Mihajlo Pupin realizovan je grafički modul baziran na jednom takvom kontroleru koji bi se ugradjivao kao opcioni dodatak u VT-100 kompatabilan CRT terminal, TIM-001.

Modul se može koristiti i univerzalnije, jer je logički prilagodljiv za ugradnju na bilo koji mikroprocesorski sistem.

Okosnicu modula predstavlja specijalizovani grafički VLSI čip - INTEL 82720 grafički displej kontroler (GDK).

2. INTEL 82720 GRAFIČKI DISPLEJ KONTROLER (GDK)

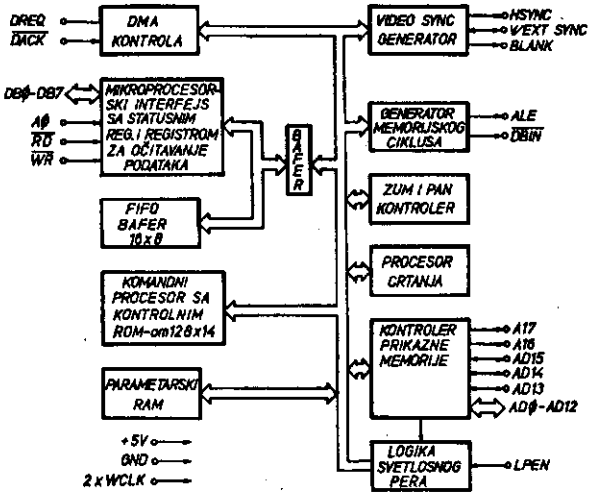
82720 GDK (sl. 1) je integrisani mikroprocesorski periferijski čip, predvidjen za rad u sprezi sa bilo kojim mikroprocesorom generalne namene.

Projektovan je tako da podržava rad rasterskih grafičkih i alfa-numeričkih displeja.

GDK odvaja sistemski procesor od prikazane memorije obavljajući funkcije neophodne za generisanje rasterske slike i kontrolu i upravljanje prikaznom memorijom.

Bazične funkcije koje obavlja GDK su:

- generisanje sinhronizacionih signala potrebnih za rad monitora;
- generisanje signala za adresiranje prikazne memorije u cilju osvežavanja ekrana;
- modifikacija prikazne memorije za vreme operacije iscrtavanja slike;
- arbitracija pristupa prikaznoj memoriji izmedju povremenih zahteva za upis sa jedne i logike za očitavanje memorije u cilju osvežavanja ekrana sa druge strane.



Sl. 1. Blok dijagram INTEL 82720 GDK-a

GDK omogućava programsko formatiranje prikaznog ekrana, kao i dužinu trajanja signala zamračivanja ("BLANKING"), horizontalnih i vertikalnih sinhro impulsa i sl.

GDK direktno podržava do 256K 16-bitnih reči prikazne memorije (512 KBY). Omogućeno je zumiranje, panovanje i skrolovanje celog ili nezavisnih delova ekrana.

Programski GDK se inicira za rad u:

- grafičkoj konfiguraciji (koja se koristi u ovom projektu),
- karakter konfiguraciji (kada radi kao kontroler alfa-numeričkih displeja) i
- kombinovanoj konfiguraciji (kada se prikazuje i slika i tekst).

U grafičkoj konfiguraciji GDK zahteva da se prikazna me-

morija organizuje u 16-bitne, odnosno 32-bitne reči, zavisno od toga u kome je režimu rada.

Naime, programski se GDK orijentira za rad u tzv. normalnom režimu rada kada pristupa memoriji kao linearnom nizu 16-bitnih reči, ili tzv. "wide display" režimu kada se za isto vreme očita 32-bitna reč.

Kontroler prikazne memorije (sadržan u GDK) generiše dva tipa memorijskih ciklusa: očitavanje (READ) prikazne memorije, kao i ciklus očitavanja - modifikacija - upisivanje (RMW) za vreme koga se prikazna memorija modifikuje radi dobijanja željene slike.

GDK raspolaže setom komandi za generisanje grafičkih primitiva što pojednostavljuje izradu softvera (firmvera) i još više rasterećuje glavni procesor.

Komande koje GDK prima se mogu podeliti na pet kategorija:

- kontrolne video komande koje specificiraju rasterske parametre (npr. broj tačaka po liniji, dužina aktivne linije, dužina trajanja sinhro signala, itd.);
- kontrolne displej komande koje iniciraju skaniranje prikazne memorije, odredjuju faktor zumiranja, podelu površine displeja, i sl;
- komande crtanja koje sadrže komande za crtanje grafičkih primitiva kao što su linije, krugovi, lukovi i pravougaonici;
- komande očitavanja podataka iz prikazne memorije;
- kontrolne DMA komande.

Sve ove mogućnosti kao i fleksibilnost GDK-a omogućile su lakše projektovanje grafičkog modula terminala TIM-001.

3. OPIS ORGANIZACIJE MODULA

Modul je realizovan na ploči formata "dupla Evropa" sa standardnom sprežnom magistralom procesora INTEL 8085, na bazi koga je realizovan CRT terminal TIM-001.

Logički sistem je prilagodljiv za ugradnju u bilo koji mikroprocesorom kontrolisani uredjaj.

Sistem je koncipiran oko grafičkog displej kontrolera (GDK) INTEL 82720.Kako se radi u crno-belom monitoru, ne postoji zahtev za bojama, ali je modu koncipiran tako da je omogućeno prikazivanje 4 nijanse sivog, bez značajnih hardverskih modifikacija.

Softver sistema omogućava korišćenje podskupa instrukcija GKS grafičkog standarda.

3.1. Hardverska organizacija

Blok dijagram hardvera koji podržava rad sa 4 nijanse sivog je prikazan na slici 2. Osnovni elementi realizovanog modula za crno-beli režim rada su:

- grafički displej kontroler 82720 sa interfejsom prema magistrali 8085 procesora;
- prikazna memorija i njen interfejs prema GDK;
- pomerački (šift) registar izmedju displej memorije i video sinhronizatora;
- sistemski kontroler.

3.1.1. Grafički displej kontroler sa interfejsom prema magistrali 8085 procesora

Grafički displej kontroler je detaljno opisan u ranijem izlaganju pa ćemo se više zadržati na interfejsu prema 8085 magistrali. GDK zauzima dve adrese u procesorskom ulazno/izlaznom prostoru.

Korišćenjem adresnog bita A0 omogućeno je razlikovanje komandi od parametra pri upisu, odnosno podataka od statusa pri čitanju.

Kako GDK nema čip selekt (CS), to se signali za upis (WR) i čitanje (RD) moraju vrednovati signalom koji selektuje grafički modul.

3.1.2. Prikazna (displej) memorija i njen interfejs prema GDK

Postojeća rezolucija CRT terminala TIM-001 u alfa režimu od 560 tačaka x 240 linija (odnosno 80 karaktera u 24 reda, pri čemu je svaki karaketr formata 7 x 10 tačaka) odredjuje rezoluciju grafičkog modula a time i potrebnu veličinu prikazne memorije.

Dodatno ograničenje uvodi i GDK sa zahtevom da horizontalna rezolucija (broj tačaka na jednoj horizontalnoj liniji) bude deljiva sa 32 [2]. Na ovaj način se rezolucija grafičkog modula svodi na 544 tačaka x 240 linija. Ovakva rezolucija zahteva 16320 bajtova ili 8160 16-bitnih reči za crno-beli odnosno dvostruko više za kolor režim (4 nijanse sivog).

Prikazna memorija je realizovana u jednoj ravni upotrebom 16 DRAM čipova 4116 kapaciteta 16K x 1 bita, što daje ukupni kapacitet od 32 KBY. Ovakvu organizaciju nameće GDK koji memoriji pristupa kao linearnom nizu tačaka organizovanih u 16-bitne reči, [1], [2].

Za postojeću realizaciju od 544 x 240 tačaka ovaj kapacitet daje dve alternative.

Prva je da se upotrebi 1 bit po tački što omogućava da jedna tačka može imati nivoe crnog i belog. U ovom slučaju potrebno je 16 KBY memorije. Preostalih 16 KBY se mogu koristiti za kadriranje ("panning", "scrolling").

GDK u ovom slučaju radi u normalnom režimu rada pristupajući memoriji kao linearnom nizu tačaka organizovanih u 16-bitne reči.

Druga alternativa je da se koriste 2 bita za informacije o svakoj tački. To nam daje mogućnost da svaka tačka može imati 4 nivoa - crnog, belog i dva nivoa sivog.

Na ovaj način svih 32 KBY prikazne memorije je iskorišćeno. GDK u ovom slučaju radi u tzv. "wide display" režimu rada, pristupajući memoriji dva puta za isto vreme za koje se u normalnom režimu rada pristupa jedanput, radeći na taj način, sa 32-bitnim rečima.

Tako se zamenjuje dve odvojene 16 KBY memorijske ravni, po jedna ravan za svaki bit po tački.

Ova druga alternativa, nameće i zahtev za dodatnim hardverom, počevši od proširenja D/A konvertora koje omogućava dobijanje 4 različita nivoa video signala, preko dodatne logike sistemskog kontrolera (videti poglavlje 3.1.4.), do dodavanja još jednog pomeračkog registra.

3.1.3. Pomerački (šift) registar

16-bitna reč se iz prikazne memorije doprema u pomerački registar sa paralelnim ulazom i serijskim izlazom.

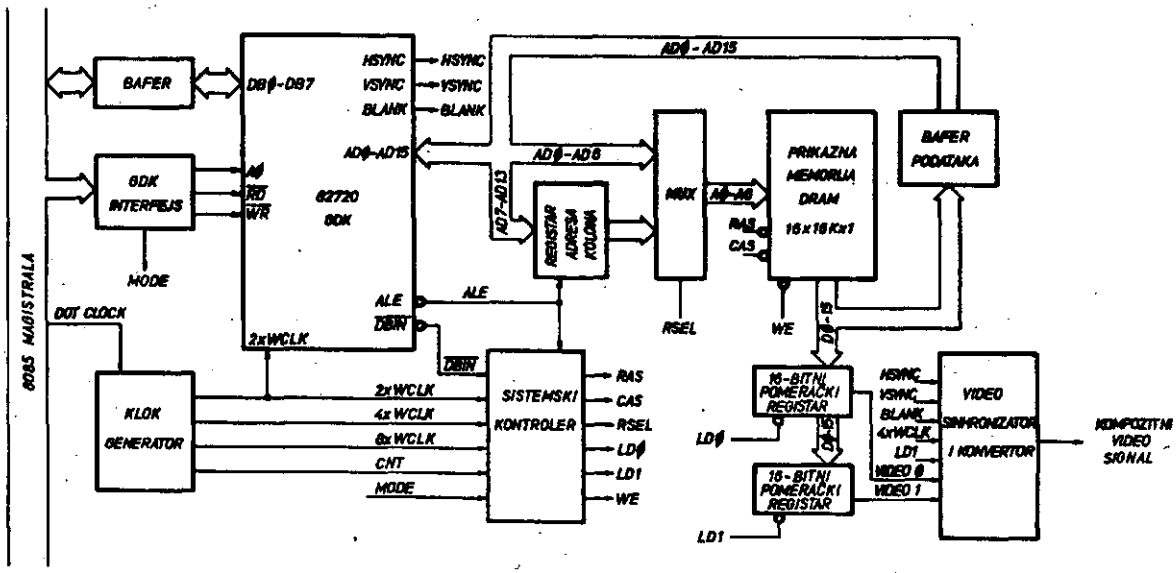
Taktom od 10,92 MHz ("dot clock") se odgovarajućim redom vrši iščitavanje pomeračkog registra.

Iščitani signali se potom dovode do video sinhronizatora koji obrazuje signal upotrebljiv za monitorski deo terminala.

Ukoliko se koriste dva bita po tački, potrebna su i dva 16-bitna pomeračka registra, po jedan za svaku memorijsku ravan. Način rada je identičan kao i kada postoji samo jedan bit po tački izuzev što je potrebno omogućiti sinhronizaciju rada pomeračkih registra, kao i dodati jedan ulaz D/A konvertora za dodatni iščitani signal iz pomeračkog registra.

3.1.4. Sistemski kontroler

Na sl. 3 prikazan je jedan od oblika kontrolnih signala neophodnih za realizovan crno-beli režim rada grafičkog modula. Odgovarajući dijagram stanja konačnog automata na osnovu koga je i isprojektovana kontrolna jedinica dat je na sl. 4.



Sl. 2. Blok dijagram grafičkog modula koji podržava rad u kolor režimu

Dijagramom nisu predvidjene reakcije automata na sve moguće kombinacije ulaznih signala (DBIN, ALE) jer pri ispravnom funkcionisanju GDK obezbedjuje njihovo ekskluzivno pojavljivanje [1]. Takodje, u cilju postavljanja što manjeg broja stanja i dodatnih kontrolnih signala sistemski kontroler je realizovan parcijalno, u tri nezavisne celine sa zajedničkim početnim stanjem. Kao sistemski klok pri generisanju RAS, CAS i RSEL signala koriste se DOT CLOCK (10,92 MHz) dok se za WE i LD1 koriste klok kontrolera (2xWCLK) i njegov komplement respektivno. Prema klasifikaciji konačnih automata ("final state machines") [4], grane I i III (sl. 4) predstavljaju Moor-ove, a grana II Mealy-ev automat za generisanje sekvenci izlaznih signala kao reakciju na odgovarajući ulazni signal ("one shot pattern generator"). Vremenska funkcija bilo kojeg izlaznog kontrolnog signala z(t) može se predstaviti relacijom

$$z(t, t+k-1) = P$$

ako je

$$x(t) = \text{start}$$

i

$$z(t) = 0$$

ako je

$$x(t-i) \neq \text{start za } i = 0,\ldots,k-1$$

gde je

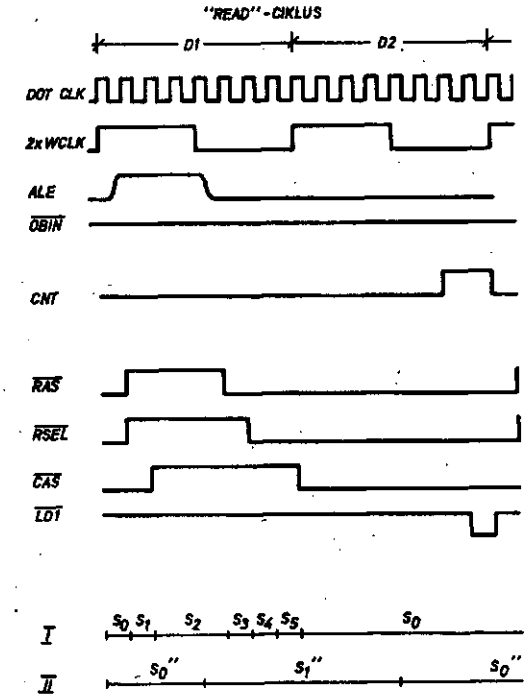
$P=(P_0,\ldots,P_{k-1})$ sekvenca dužine k osnovnih sistemskih klokova a

$x(t)$ pobudni ulazni signal (ALE,DBIN).

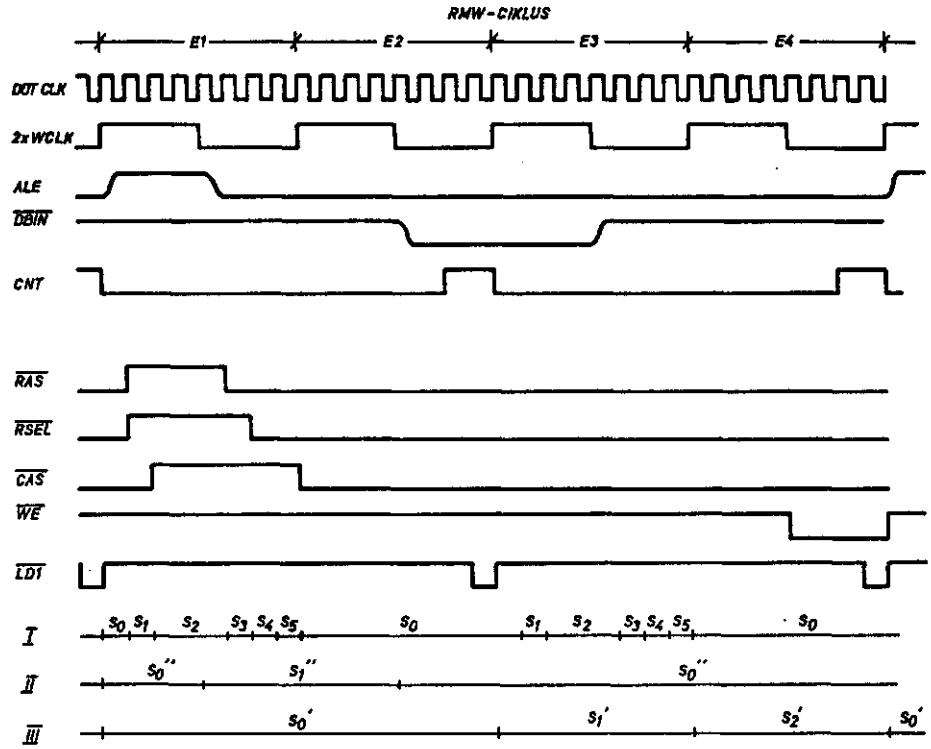
Pomoćni ulazni signal CNT se formira radi tačnog pojavljivanja LD1 signala na samom kraju ciklusa čitanja prikazne memorije. Kontroler je realizovan sa 5 standardnih TTL kola serije 74.

Kako je već pomenuto, u konačnoj verziji modul će podržavati rad sa 4 nijanse sivog osvetljaja. Oblik kontrolnih signala kojima se obezbedjuje ovakav režim rada dat je na sl. 5. Signali LD1 i WE generišu se kao u crno-belom režimu rada, što znači da, u odnosu na postojeću verziju, treba preprojektovati deo sistemskog kontrolera definisanog granom I na sl. 4. Pri tome, osnovni problem se sastoji u činjenici da sada istom memorijskom čipu u toku jednog ciklusa treba pristupiti dva puta. Kako ciklus čitanja traje oko 1,5 uS to je dovoljno vremena da se u "RIPPLE" modu [5] očitaju dve memorijske lokacije čije se adrese razlikuju samo u bitu najvece težine (A13).

Smeštanje očitanih podataka u pomeračke registre vrši se u različitim trenucima, definisanim signalima LD0 i LD1. Razlika se kasnije kompenzuje odgovarajućim kašnjenjem video signala 0 u video sinhronizatoru.



Sl. 3a Vremenski dijagram ulaznih i izlaznih kontrolnih signala "READ" ciklusa za crno-beli režim rada



Sl. 3b Vremenski dijagram ulaznih i izlaznih kontrolnih signala "RMW" ciklusa za crno-beli režim rada

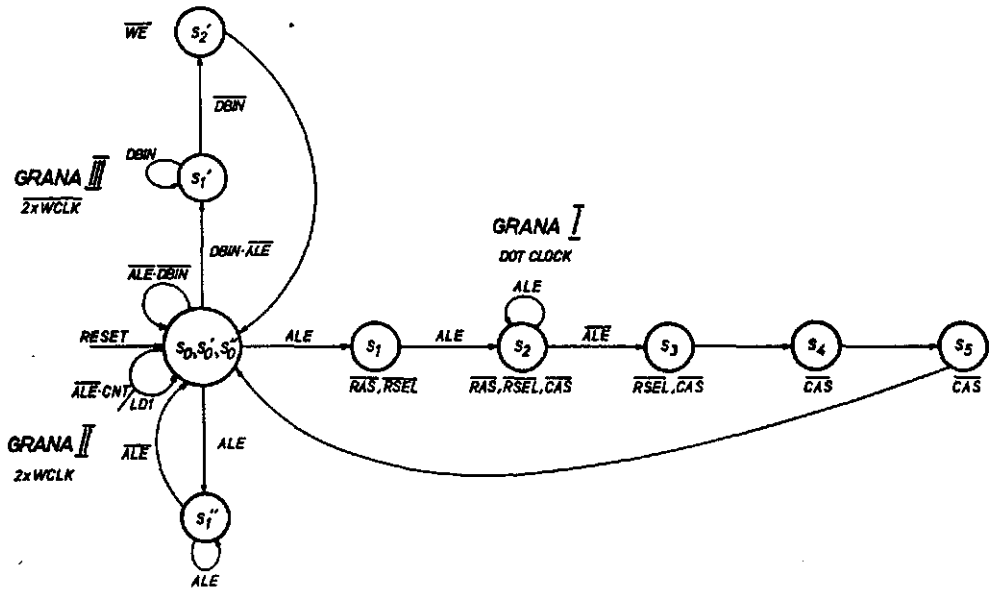
3.2. Softverska podrška

Predvidja se implementacija OA nivoa GKS grafičkog standarda. Zbog male procesorske moći terminala sav softver koji se korisniku stavlja na raspolaganje biće podeljen na dva dela:

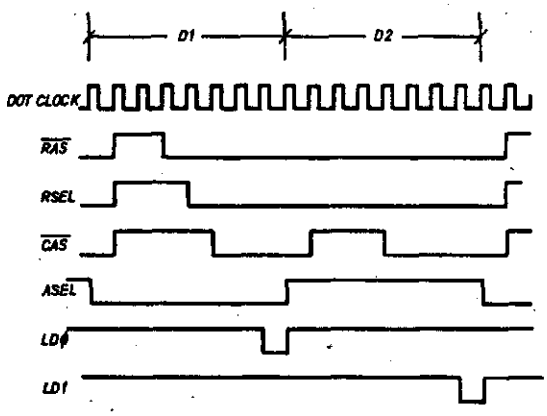
- jezgro grafičkog sistema u samom terminalu i
- grafički programski paket na "host" računaru.

Od svih funkcija koje treba obezbediti pomenuti nivo GKS-a na "host" računaru se obavlja normalizacija koordinata i odsecanje ("clipping") u odnosu na specificirano vidno polje ("viewport").

Detaljniji prikaz softverske podrške dat je u posebnom radu [3].



Sl. 4. Dijagram stanja sistemskog kontrolera



Sl. 5. Vremenski dijagram kontrolnih signala kojima se podržava rad u kolor modu

4. ZAKLJUČAK

Uz pomoć VLSI grafičkog displej kontrolera realizovan je grafički modul na jednoj ploži formata "dupla Evropa" koji podržava rad u crno-belom režimu rada namenjen ugradnji u CRT terminal TIM-001. Dodavanjem ovog modula znatno se proširuju upotrebne mogućnosti terminala.

Modul je zbog ograničene rezolucije i rada u crno-belom režimu pogodan za CAD/CAM aplikacije nižeg nivoa složenosti kao i za upotrebu u poslovnim sistemima.

U toku je rad na realizaciji kolor verzije modula (verzije sa 4 nijanse sivog).

Na bazi istog kontrolera i slične organizacije pod kontrolom Intel-ovog mikroprocesora 80186 u Institutu Mihajlo Pupin projektuje se kolor terminal visokog kvaliteta za primene u CAD/CAM sistemima. Terminal će imati rezoluciju od 1024 x 1024 prikaznih tačaka sa mogućnošću istovremenog prikazivanja 16 različitih iz palete od 4096 boja. Softver će obezbedjivati interaktivni rad i sve funkcije predvidjene nivoom 1b GKS standarda.

5. REFERENCE

1. INTEL 82720 GRAPHICS DISPLAY CONTROLLER DATA SHEET, 1983, USA.

2. INTEL 82720 GDC APPLICATION MANUAL, July '83, USA.

3. "Grafički softver za terminal TIM-001 prema GKS standardu", Vraneš P., Bajčetić M., INFORMATICA '85.

4. "DIGITAL SYSTEMS AND HARDWARE/FIRMWARE ALGORITHMS", Miloš D. Ercegovac, Tomash Lang, John Wiley a Sons, 1985, USA.

5. SEMICONDUCTOR INTEGRATED MICROCIRCUITS K 565 PY 3 REFERENCE DATA, ELERS, MOSCOW, USSR.

ARHITEKTURA INTELIGENTNOG 2D GRAFIČKOG TERMINALA VISOKE REZOLUCIJE

Ljiljana Damnjanović, Sanja Bilčar, Dušan Marinčić,
Zoran Konstantinović
Institut Mihajlo Pupin, Volgina 15, Beograd, Yugoslavia

UDK: 681.3.014

SADRŽAJ - Opisana je arhitektura dvodimenzionalnog (2D) rasterskog grafičkog terminala visoke rezolucije koji je zasnovan na MULTIBUS kompatibilnim računarskim karticama. Grafički sistem se koristi u konfiguraciji za razvoj i "host"--terminal konfiguraciji. U konfiguraciji za razvoj koriste se mogućnosti instaliranog operativnog sistema, dok se "host"-terminal konfiguracija oslanja na realizovanu osnovnu softversku podršku (OSP) grafičkog sistema. Opisane su takodje karakteristike grafičkog softvera (Rasterski grafički paket - RGP) sa realizovanim grafičkim komandama.

AN ARCHITECTURE OF INTELIGENT HIGH RESOLUTION 2D GRAPHICS TERMINAL

ABSTRACT - A two dimensional (2D) high resolution raster graphics terminal based on a Multibus-compatible computer cards has been implemented. The terminal can be configured in two ways: as a stand-alone system and in host-terminal version. In the stand-alone configuration the installed operating system is used while in the host-terminal configuration the basic software package (OSP) is implemented. The characteristics of graphics software (Raster graphics package - RGP) with graphics commands have been presented.

1. UVOD

Veliki broj savremenih računarskih aplikacija koristi grafičku predstavu u cilju poboljšanja kvaliteta interakcije čovek-mašina.Mogućnost vizuelnog praćenja toka programa uz interaktivni dijalog, korekcije delova kompleksne slike, pamćenja i multipliciranja elemenata slike, itd., čine računarski sistem veoma primamljivim sa tačke gledišta korisnika. Grafički sistemi visoke rezolucije su do sada bili veoma skupe komponente računarskih konfiguracija jer su bili zasnovani na skupim vektorskim pokazivačkim elementima. Povećanjem gustine pakovanja memorijskih komponenata i drastičnim padom njihove jedinične cene, jeftiniji rasterski grafički sistemi postaju sve popularniji. Kod njih su uklonjeni osnovni nedostaci u pogledu rezolucije i memorije (lit /1/, /2/, /3/, /6/ tako da su do izražaja došle dobre strane ovih sistema:

- veliki broj boja;
- proizvoljna kompleksnost slike;
- mogućnost popunjavanja delova slike;
- nezavisnost nastanka treptanja (flicker) od kompleksnosti slike;
- prikaz više nezavisnih slika na istom ekranu;
- pristupačna cena.

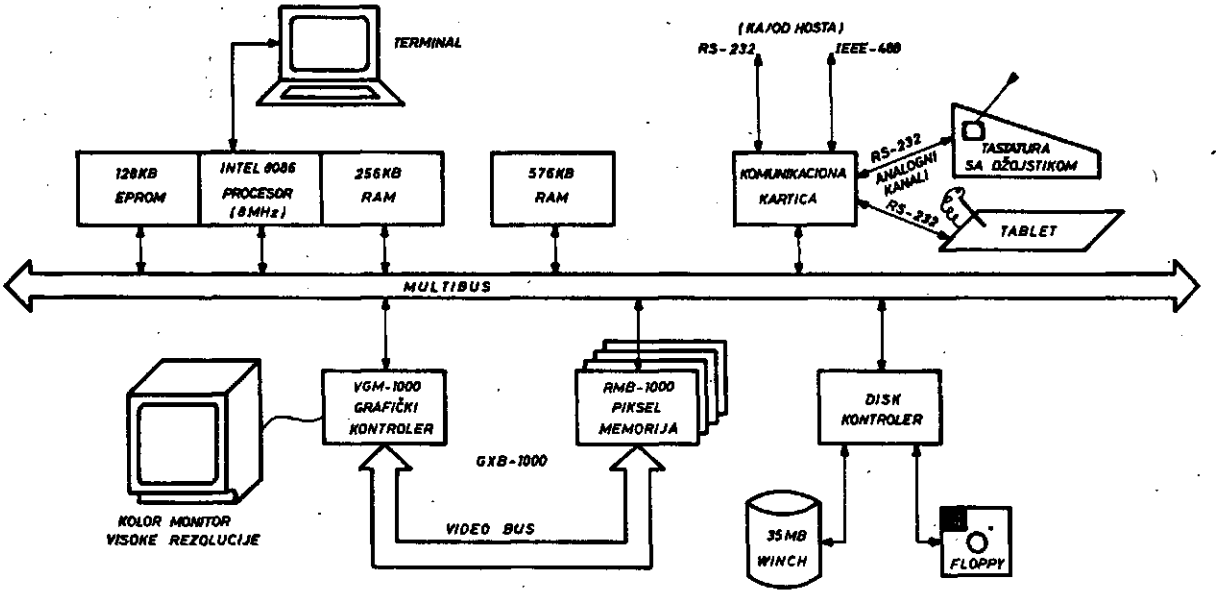
U ovom radu opisana je arhitektura jednog rasterskog sistema visoke rezolucije. U glavi 2 data je generalna blok šema hardverske organizacije sistema. Osnovne karakteristike date arhitekture su zasnovanost na standardnim komercijalno raspoloživim MULTIBUS kompatabilnim računarskim karticama; fleksibilnost pri definisanju komponenata konkretnog terminala. Glava 3 daje dva primera konfiguracije grafičkog sistema: konfiguracija za razvoj i "host" - terminal konfiguracije. U konfiguraciji za razvoj koriste se mogućnosti instaliranog operativnog sistema, dok se "host" - terminal konfiguracija oslanja na osnovnu softversku podršku (OSP) grafičkog sistema.

Na kraju su opisane karakteristike razvijenog grafičkog softvera (Rasterski grafički paket - RGP) sa realizovanim grafičkim komandama.

2. HARDVERSKA KONFIGURACIJA SISTEMA

Na sl. 1 vidi se konfiguracija inteligentnog grafičkog terminala. Osnova sistema je rasterski grafički kontroler GXB-1000 (proizvodjač Matrox, Kanada) visoke rezolucije (1024 x 768 poluslike) u boji (moguće je prikazivati istovremeno 256 boja).

Kao što se vidi na sl. 1 (GXB-1000 se sastoji od dve kartice koje povezuje VIDEO BUS. To su VGM-1000 - grafički kontroler i RMB-1000 piksel memorija. Kako VGM-1000 ima snažnu procesorsku podršku (četiri "pipelined" procesora) moguća je brzina crtanja vektora od 800 nsec/piksel, zatim crtanje 2D osnovnih grafičkih primitiva i segmentirana struktura slike. Standardna veličina jedne piksel memorije je 1024 x 1024 piksela sa 4 bita/pikselu što čini 512 Kb. U punoj konfiguraciji grafički kontroler ide sa četiri piksel memorije koje se mogu povezati paralelno, redno ili mešovito. Pored osnovnih grafičkih primitiva (crtanje tačke, kruga, po-



Slika 1. HARDVERSKA KONFIGURACIJA SISTEMA

ligona, popunjavanje elemenata, definisanje atributa (crtanja,itd.), kontroler podržava i programsko definisanje nezavisnih slika, podešavanje formata rastera, direktne operacije nad piksel memorijom, dijaloške komunikacione sekcije i ulazne funkcije.

Komunikacije izmedju korisničkog procesora (Intel 8086) i grafičkog kontrolera odvija se na dva načina:

1. Preko U/I porta čiju adresu korisnik specificira.
2. Posredstvom MULTIBUS magistrale kojom korisnički procesor i kontroler imaju pristup korisničkoj RAM memoriji.

Preko U/I porta kontroler dobija samo pet osnovnih komandi:

- zadavanje adrese display programa u RAM-u;
- reset;
- čitanje statusa kontrolera;
- stop;
- start.

Po dobijanju startne adrese display programa i komande preko U/I porta, kontroler preko MULTIBUS magistrale pristupa zadatoj adresi u RAM memoriji i počinje izvršavanje komandi display programa. Display program je prethodno kreiran od strane korisničkog procesora, a sadrži grafičke komande koje interpretira i izvršava kontroler. Po izvršenju display programa, kontroler generiše prekid kojim naznačuje korisničkom procesoru da je završio /7/.

Na ovaj način moguće je ostvariti paralelizam rada procesora 8086 i grafičkog kontrolera /4/.

Glavni procesor sistema je Intel 8086 i on se koristi za izvršavanje programa kojima se realizuju 2D grafičke karakteristike, segmentirana struktura slike, komunikacija sa "host" računarom, interaktivni dijalog i sinhronizacija sa grafičkim kontrolerom. Za sada procesor radi sa 1 Mb memorije.

Na komunikacionoj kartici postoje dva RS-232 asinhrona serijska kanala čija je brzina do 9600 boda, zatim jedan paralelni IEEE-488 DMA kanal sa brzinom do 300 Kb/sec i A/D ulazi za podršku džojstika. Kao što se na sl. 1 vidi preko ove kartice ostvarena je veza sa host računarom i ulaznim uredjajima (tastatura sa džojstikom i tablet).

Preko kartice disk kontrolera priključeni su vinčester disk kapaciteta 35 MB i flopi disk.

Opisani sistem čija je arhitektura prikazana na sl. 1 može da se konfiguriše u zavisnosti od potreba korisnika kao terminal nekog "host" računara ili kao zasebna grafička stanica za razvoj.

3. KONFIGURACIJE GRAFIČKOG TERMINALA

Dve osnovne konfiguracije grafičkog terminala visoke rezolucije. To su univerzalni, mikroprocesorski razvojni sistemi sa grafičkim mogućnostima i tzv. "host"-terminal konfiguracija kod koje se grafički terminal sa znatno

skromnijim mogućnostima u odnosu na prvu varijantu vezuje za proizvoljan "host" računar.

3.1. Razvojni sistem sa grafičkom podrškom

Hardverska blok šema na sl. 1 predstavlja konglomerat komponenti obeju konfiguracionih varijanti. Uz standardan razvojni alat koji podrazumeva:

- operativni sistem (XENIX ili RMX 86);
- programske prevodioce (FORTRAN, PASCAL, C, PL/M-86);
- uslužne programe za razvoj (EDITOR, BIBLIOTEKAR, LINKER, LOKATOR...).

Ovaj mikroprocesorski razvojni sistem ima snažnu podršku. Rasterski grafički paket (RGP) rezidira u 128 KB EPROM-u. Korisnička grafička aplikacija koristi RGP, linkujući se sa bibliotekom pozivnih procedura paketa (GRAF.LIB). Ova biblioteka veoma malo opterećuje aplikaciju, jer pozivna procedura predstavlja samo indirektno aktiviranje (pomoću adrese) ROM rezidentne grafičke primitive. Ovako definisanom grafičkom podrškom može se veoma jednostavno ekspandovati gotovo proizvoljan mikroprocesorski razvojni sistem. Sva intervencija sastoji se u dodavanju RGP EPROM-a na procesorsku ploču. Naravno, podrazumeva se da sistem ima dovoljno RAM memorije potrebne grafičkom paketu.

Razvojni sistem može imati višestruku namenu - za razvoj opštih programa, za razvoj same grafičke podrške, za razvoj grafičkih aplikacija koji koriste RGP kao i aplikacija koje će se eksploatisati na "host"/terminal varijanti.

3.2. "Host" - terminal konfiguracija

Hardverska konfiguracija ove varijante grafičkog terminala dobija se kad se sa sl. 1 isključi disk kontroler. Terminal se sa "host" računarom vezuje standardnom serijskom i/ili paralelnom linijom. Variranjem količine RAM memorije može se inkrementalno podešavati grafička snaga terminala. Ovakvom konfiguracijom postiže se da, bez obzira na kompleksnost grafičke aplikacije ona predstavlja veoma malo memorijsko i procesorsko opterećenje hosta.

U ROM memoriji terminala se pored monitorskog programa koji sadrži elementarni "debugger" (opciono) i grafičkog paketa (RGP identičan onom u razvojnoj varijanti) nalazi i osnovna sistemska podrška (OSP). Dve su osnovne funkcije OSP-a:

- komunikacija grafičkog terminala sa "host" računarom i U/I uredjajima;
- upravljanje radom RGP-a ("long term scheduling").

OSP obezbedjuje prioritetnu obradu asinhronih dogadjaja vezanih za ulazno/izlazne uredjaje. Procesi koji ove dogadjaje opisuju (ustvari prekidne procedure koje pune/prazne FIFO bafere) predstavlja "foreground" režim u kom se procesor obično vrlo kratko zadržava. Na ovaj način baferisani podaci sa hosta i interaktivnih ulazno/izlaznih uredjaja obradjuju se u tzv. "backround" režimu. Dodela procesora "background" procesima, odnosno sinrhonizacija izvršavanja grafičkih komandi od strane RGP-a, vrši se na bazi prioriteta. Inicijalno, sve funkcije imaju predefinisane fiksne prioritete. Da bi se izbegla kontencija na resursnim baferima koja bi mogla nastati kao rezultat neuniformnog pristizanja podataka sa hosta i različitih ulazno/izlaznih uredjaja, obezbedjena je i mogućnost dinamičke promene prioriteta "backround" procesa. Prioritet procesa podiže se proporcionalno brzini kojom pristižu podaci u bafer koji taj proces obradjuje. Ovakvim balansiranjem obrade izbegava se mogućnost zagušenja na pojedinim kanalima.

Na proizvoljnom "host" računaru implementira se biblioteka pozivnih grafičkih procedura i biblioteka komunikacionih procedura.

Biblioteka pozivnih grafičkih procedura (GRAF.LIB) identična je onoj na razvojnoj varijanti terminala. Potpuno je portabilna jer je realizovana u standardnom FORTRAN 77 jeziku a pretpostavlja se da svaki "host" ima prevodilac za ovaj jezik. Ne postavlja se pitanje efikasnosti ovih procedura jer im je funkcija trivijalna - prenos kontrole na odgovarajuće grafičke primitive u ROM-u.

Biblioteka komunikacionih procedura (COM.LIB) je delimično neportabilna jer odražava karakteristike ulazno/izlaznog podsistema operativnog sistema konkretnog računara. Biblioteka sadrži procedure kojeopslužuju serijsku i/ili paralelnu liniju kojom "host" komunicira sa grafičkim terminalom (slanje komandi, prihvatanje odgovora,...). Da bi bile što efikasnije, ove procedure obično sadrže U/I sistemske pozive niskog nivoa koji najviše zavise od mašine. Ova neportabilnost se procenjuje na oko 20%.

4. SOFTVERSKI PAKET GRAFIČKOG TERMINALA - RGP

RGP je projektovana kao grafički softverski paket opšte namene koji povezuje aplikacioni program i grafički hardver. Obrada zadatog grafičkog modela je nezavisna od grafičkog kontrolera. Na kraju kada je slika generisana u meta displej fajlu /4/, /5/ na osnovu internih komandi i sračunatih realnih displej koordinata grafičkih objekata, generišu se grafičke komande konkretnog grafičkog kontrolera. Tako se postiže visoka portabilnost RGP softvera.

RGP podržava transformacije vidnog polja, geometrijske transformacije, odsecanje nevidljivih objekata, segmentnu

strukturu slike i manipulaciju sa segmentima i displej fajlom. Grafički model se zadaje RGP grafičkim komandama u koordinatama okoline. Na osnovu podataka o vidnom polju koje se zadaje jednom od grafičkih komandi vrši se normalizacija koordinata i rezultat se smešta u meta displej fajl. Na osnovu zadatih geometrijskih transformacija pomoću koje se vrši 2D geometrijska transformacija grafičkih objekata.

Posle toga sledi odsecanje (clipping) nevidljivih objekata van ivica polja (0,0; 1,1) normalizovanog sistema. Poslednja transformacija je preslikavanje u displej koordinate na osnovu zadate sekcije terminala za prikazivanje slike. Na osnovu formiranog meta fajla generiše se displej fajl sa grafičkim komandama konkretnog grafičkog kontrolera. Izvršavanjem ovog displej fajla (programa) od strane grafičkog kontrolera dobija se slika na ekranu monitora.

Grafičke komande realizovane paketom RGP koriste se pomoću skupa pozivnih procedura definisanih grafičkom bibliotekom.

Grafička biblioteka je skup od 36 procedura izabranih po ugledu na CORE sistem koji su prihvatili mnogi proizvodjači kao standard i iz koga je proistekao GKS (Graphics Kernal System) standard ustanovljen nedavno.

RGP grafičke komande su:

AINT	Alfanumerički mod
ARC	Crtanje kružnog luka
CFILL	Popunjavanje kruga
CIRCLE	Crtanje kruga
COLR	Zadavanje indeksa linije
CREATE	Kreiranje segmenta
DCLEAR	Brisanje dijaloške sekcije
DELETE	Brisanje segmenta
DRAW	Crtanje linije
EFILL	Kraj popunjavanja poligona
ENBDLG	Kreiranje dijaloške sekcije
ENDS	Kraj segmenta
FSCR	Pomeranje naviše
GSIZ	Zadavanje veličine teksta
LSTL	Zadavanje vrste linije
MARK	Crtanje markera
MOVE	Pomeranje pera
PGDD	Popunjavanje četvorougla
PILS	Zadavanje vrste šrafiranja
POLY	Crtanje poligona
RDRAW	Crtanje linije - relativno
RMOVE	Pomeranje pera - relativno
RSCR	Pomeranje naniže
SCROLL	Zadavanje pomeranja
SDACHR	Zadavanje karaktera u dijaloške sekcije
SDAIND	Zadavanje indeksa dijaloške sekcije
SDAPOS	Zadavanje pozicije dijaloške sekcije
SETBCD	Zadavanje boje pozadine
SETGSL	Zadavanje nagiba teksta
SETMRT	Zadavanje tipa markera
SETTIN	Zadavanje indeksa teksta
SFILL	Početak popunjavanja poligona
TEXT	Crtanje teksta
TRANSM	Zadavanje pozicije teksta
VISIBL	Zadavanje vidljivosti segmenta
WIND	Zadavanje vidnog polja

Proširenje asortimana komandi brojčano nije limitirano, a samo njihovo definisanje je veoma jednostavno i ne iziskuje komplikovane softverske intervencije.

5. ZAKLJUČAK

Grafički terminal opisan u ovom radu realizovan je na razvojnom sistemu Intel 86/330 i do sada implementiran u "host" - terminal konfiguracijama na šasijama MATROX CCB-7 i CCB-9. Kao "host" računari korišćeni su PDP-11/70, PDP-11/34, Intel 286/310 (serijska veza 9600 boda) i HP-1000 (serijska veza 9600 boda i paralelna IEEE-488). Terminali su korišćeni u aplikacijama prikazivanja dinamičkih objekata na kompleksnoj pozadini uz interaktivnu kontrolu i tu se pokazali veoma dobro. Dalji rad na grafičkom terminalu biće usmeren ka povećanu rezolucije (limit grafičkog kontrolera), proširenju skupa grafičkih komandi po ugledu na standard GKS i uvodjenju treće dimenzije.

LITERATURA

/1/ W.M. Newman, R.F. Sproull, Principles of Interactive Computer Graphics, New York, McGraw - Hill 1982.

/2/ J.D. Foley, A. Van Dam, Fundamentals of Interactive Computer Graphics, Mass. Addison - Wesley, 1982.

/3/ D.J. Doornink, J.C.Dalrymple, "The Architectural Evolution of a High-Performance Graphics Terminal", IEEE Computer Graphics and Applications, April, 1984.

/4/ Z. Konstantinović, R. Stanković, Lj. Damnjanović, 2D raster grafički terminal visoke rezolucije, Zbornik materijala XXIX Jugoslovenske konferencije ETAN, Niš, Jun, 1985.

/5/ S. Bilčar, Z. Konstantinović, Baza podataka 2D raster grafičkog terminala visoke rezolucije, Zbornik materijala XXIX Jugoslovenske konferencije ETAN, Niš, Jun, 1985.

/6/ G.I. Wsham, Design trade-offs in a multifunction graphics terminal MINI-MICRO Systems, Nov, 1983.

/7/ Matrox electronics systems ltd, GXB-1000 Virtual Graphics Machine Manual, 186-A50-02/0 released: 8/12/83.

BARVE V RAČUNALNIŠKI GRAFIKI

M.Kastelic,B.Grilec,P.Peterlin,ISKRA-Avtomatika,Ljubljana,
Jugoslavija

UDK: 681.326

POVZETEK - V tem sestavku sem obdelal problem tvorjenja napetostnih nivojev za barvni slikovni zaslon na osnovi CIE diagrama. Nakazal sem tudi, kje so glavni problemi. Osnova pri vsem tem pa mi je sistem, sestavljen iz enot GVK (grafični video krmilnik) in GVR (grafični video pomnilnik), ki smo ga izdelali v Iskri-Avtomatiki.

ABSTRACT - In this short paper I treated the problem of voltage level formation for the color VDU on the basis of CIE diagram. I also indicated same major problems. As an overall basis, I used the system made of GVK units (graphic video controller) and of GVR (graphic video memory) which was made in Iskra Avtomatika.

1. UVOD

Barvna računalniška grafika je že bila predstavljena v mnogih člankih. V vseh teh člankih pa je zelo malo napisanega o ozadju, kako pridemo do barv, ki jih vidimo na barvnem zaslonu. Ne smemo pa pozabiti na dejstvo, da vsak človek zaznava barve na svoj način in tako lahko pride do zelo različnih gledanj na določeno barvo ali pa na vrsto barvnih kombinacij. Pomembno je tudi to, da vsako oko ne loči enako število barv in tako lahko pride do tega, da se nekomu zdita dva barvna odtenka popolnoma enaka, nekdo drug pa bo ta dva odtenka ocenil kot različna.

2. SPLOŠNO O BARVAH

Do vseh obstoječih barv lahko pridemo s pomočjo mešanja treh osnovnih barv. Izkazalo se je, da obstajata dva načina mešanja barv in tudi dve skupini osnovnih barv. Primaren ali aditiven način mešanja barv ima za osnovo naslednje tri barve: rdečo, modro in zeleno. To so takoimenovane primarne barve. Sekundarne barve, ki so osnova sekundarnemu ali subtraktivnemu načinu mešanja barv, pa so sledeče: rumena, turkizna in vijoličasta. Ta način mešanja barv se izvaja s pomočjo mešanja barvil in se uporablja na primer v tisku. O primarnem ali aditivnem načinu mešanja govorimo takrat, ko istočasno osvetljujemo določeno ploskev z dvema ali večimi barvnimi svetlobami, ali pa so majhne barvne ploskve tako blizu skupaj, da jih oko ne loči in jih vidimo kot eno ploskev. Ta način se uporablja na primer v televiziji.

Vse barve, ki jih človeško oko zazna, so zajete v barvnem telesu. Vseh teh barv ne moremo realizirati ne s primarnim načinom mešanja, kot tudi ne s sekundarnim načinom ob uporabi barvnega zaslona ali barvnega tiskalnika. Barvni tiskalnik ima še nekoliko manjši barvni obseg kot zaslon. Barvni zaslon ne more prikazati vseh barv barvnega telesa, ker ne moremo realizirati takih osnovnih barv zaslona, kot so osnova barvnega telesa. Vsako barvo je mogoče natančno opisati s tremi parametri: svetlost, nasičenost in barvni ton. Te parametre pa lahko pretvorimo v drugo trojico parametrov: R,G in B. S temi parametri lahko ravno tako natančno opišemo neko barvo kot s prvo trojico. Na osnovi R,G in B parametrov je zasnovan CIE diagram. Ker je CIE diagram dvodimenzionalen, opisati pa moramo tri parametre, je sestavljen tako, da je tretja komponenta (Z) enaka Z=1-X-Y. Za vse barve CIE diagrama je značilno, da imajo enako svetlost, razlikujejo pa se po nasičenosti barve in barvnem tonu.

3. SESTAVA SISTEMA BARVNE GRAFIKE

Sistem sodobne barvne grafike je sestavljen iz računalnika, tipkovnice, barvnega zaslona, digitalizirne tablice, sledilne krogle in svetlobnega peresa. S stališča barvne grafike sta najpomembnejša računalnik in barvni zaslon. Ostale komponente le omogočajo ali olajšujejo delo s takim sistemom. Računalnik mora imeti vgrajen grafični slikovni krmilnik in grafični slikovni pomnilnik, ki pa mora biti ločen od delovnega pomnilnika. Slikovni pomnilnik ne more biti tudi delovni pomnilnik. Grafični slikovni pomnilnik čita podatke v slikovnem pomnilniku, jih preoblikuje in posreduje barvnemu zaslonu, ki to prikaže. Grafični krmilnik poleg te funkcije opravlja še komunikacijo s centralnim računalnikom, ki mu pošilja nove podatke za slikovni pomnilnik. Te podatke mora krmilnik v času, ko ne obnavlja slike na zaslonu, vpisati v slikovni pomnilnik. Teh podatkov ne more vpisovati sam centralni računalnik, ker ne obstaja pomnilnik, ki bi ga lahko nadzorovali dve inteligentni enoti kot sta centralni računalnik in grafični krmilnik.

V grafičnem slikovnem pomnilniku morajo biti shranjeni vsi podatki o svetlosti, nasičenosti in barvnem tonu za vsako točko, ki naj bo prikazana na zaslonu. Za sestavo dobre slike na zaslonu rabimo zelo veliko število točk. Naš sistem prikazuje 640 x 480 točk, lahko pa to število tudi spreminjamo. Če bi imeli na razpolago slikovni pomnilnik z zmogljivostjo 640 x 480 bitov, bi v našem primeru lahko prikazali le črno-belo sliko. Vsak bit lahko zavzame vrednost Ø ali 1, zato je vsaka točka lahko le črna ali bela. Da pa lahko prikažemo sliko v večih barvah, rabimo ustrezno večji pomnilnik. Za prikaz slike v n barvah rabimo log n/log 2 število bitov, oziroma z m biti lahko realiziramo sliko v 2^mbarvah. Ta omejitev izhaja iz obsežnosti naslovnega področja z določenim številom bitov.

4. DELOVANJE GRAFIČNEGA ZASLONA

Enota, ki resnično prikaže sliko, je grafični barvni zaslon. Ta rabi za svoje pravilno delovanje naslednje signale: R,G, B in sinhronizacijo. Vse te signale dobi od krmilnika grafičnega zaslona. S sinhronizacijo se ureja časovni potek dela, ostali trije pa so informacije o barvah. S signali R, G in B krmilimo elektronske topove, ki obstreljujejo fosforni zaslon. Ta zaslon je sestavljen iz treh vrst fosforja. Vsak tip fosforja ob osvetlitvi oddaja drugačno valovno dolžino in s tem vzbudi v očesu občutek drugačne barve. Ker pa so ploskvice tega različnega fosforja tako blizu skupaj, da jih oko ne loči med sabo, pride do aditivnega mešanja barv. Barvna zmogljivost zaslona pa ni odvisna le od barvnega tona osnovnih barv, ampak tudi od svetlosti in nasičenosti teh barv. Vsi ti barvni parametri niso enaki kot pri osnovnih barvah CIE diagrama, zato barvni zaslon ne more prikazati vseh barv tega diagrama. Te parametre poda vsak proizvajalec za vsak tip barvnega zaslona v obliki koordinat posameznih barv v CIE diagramu. Če te tri točke vnesemo v CIE diagram in jih povežemo, dobimo trikotnik, ki nam prikazuje barvno zmogljivost zaslona.
Krmilne signale R, G in B predstavlja spreminjanje napetostnih nivojev na vhodih barvnega monitorja. S temi napetostnimi signali spreminjamo jakost elektronskih žarkov, ki osvetljujejo barvni zaslon. Te spremembe pa imajo za posledico spreminjanje svetilnosti posameznih ploskvic na zaslonu. Ker v danem trenutku istočasno osvetljujemo tri različne ploskvice zaslona, vpliva sprememba svetlosti posameznih ploskvic na spremembo barve celotne ploskve. Iz tega sledi, da moramo za dosego določene barve nastaviti točno določeno kombinacijo napetostnih nivojev na vhodih R, G in B. Te napetostne nivoje nastavlja grafični krmilnik.

5. LOOK UP TABELA IN D/A PRETVORNIK

Grafični slikovni pomnilnik čita istočasno več bitov iz slikovnega pomnilnika. V našem primeru čita od 3 do 12 bitov. S temi biti lahko preko LOOK UP tabele in D/A pretvornika nastavi določeno kombinacijo napetostnih nivojev na vhodih R, G in B v grafični zaslon. Kot vidimo, imata pri tvorjenju barv odločilno vlogo LOOK UP tabela in D/A pretvornik. V LOOK UP tabeli se biti, ki pridejo iz slikovnega pomnilnika, ustrezno prekodirajo in v D/A pretvorniku se pretvori digitalna informacija v analogno napetost. V LOOK UP tabelo so vpisani vsi parametri za vsako barvo, ki jo želimo prikazati na zaslonu. LOOK UP tabelo v našem primeru predstavlja PROM, v katerega vpišemo potrebne podatke. D/A pretvornik pa je sestavljen iz uporovnega delilnika. Ravno D/A pretvornik je tisti element, ki poleg barvnega zaslona najbolj vpliva na število barv, ki jih lahko prikažemo na zaslonu. Ker je razlika napetostnih nivojev med belo in črno barvo le $1V_{pp}$, je zelo težko realizirati veliko število napetostnih stopenj v tem območju. Ne smemo pa posabiti na veliko dinamičnost teh signalov, saj je frek-

venca v našem sistemu lahko tudi 25 MHz. Naslednja omejitev, ki nastopa v našem sistemu, je ta, da ima PROM le osem izhodnih linij. Iz tega izhaja omejitev, da lahko realiziramo le 256 barv. To se da odpraviti s tem, da vzamemo tri LOOK UP tabele, za vsako barvo svojo, vendar tega zaradi prostorske stiske in časovnih problemov nismo storili.

LOOK UP tabela nam omogoča tudi to, da nekateri biti, ki jih dobimo iz slikovnega pomnilnika, pomenijo utrip točke. Ti biti lahko nosijo tudi informacijo o tem, kje se nahaja kurzor. Najbistvenejša lastnost LOOK UP tabele je ta, da nam omogoča prikaz večih slik istočasno, ki imajo različno prioriteto. Ta prioriteta je zakodirana v LOOK UP tabeli. Prednost prioritetne razdelitve slik se pokaže pri brisanju oziroma odstranjevanju določenega dela slike iz prikaza. Pri neprioritetni razdelitvi slik bi ob brisanju ene slike odstranili na presečiščih obeh slik tudi delčke iz druge slike in na teh mestih bi ostalo črno polje. Slika ne bi bila več popolna. Z razdelitvijo slik po prioritetah in z ustrezno organizacijo LOOK UP tabele se izognemo tem nevšečnostim. Prednost je tudi ta, da na presečiščih slik prevlada barva slike z najvišjo prioriteto, ob drugačni organiziranosti pa bi tu prišlo do spremembe barv. Tu bi se pojavile neke druge barve.

Do organizacije LOOK UP tabele za delovanje v načinu prekrivanja pridemo na sledeč način. Če imamo na razpolago 12 bitov, ki jih dobimo iz slikovnega pomnilnika, to število bitov lahko poljubno razdelimo. Na primer, da želimo tri prioritetne ravnine, razdelimo teh 12 bitov med te tri prioritetne ravnine na naslednji način: površini z najnižjo prioriteto oziroma prvi ravnini naj pripadajo trije biti,

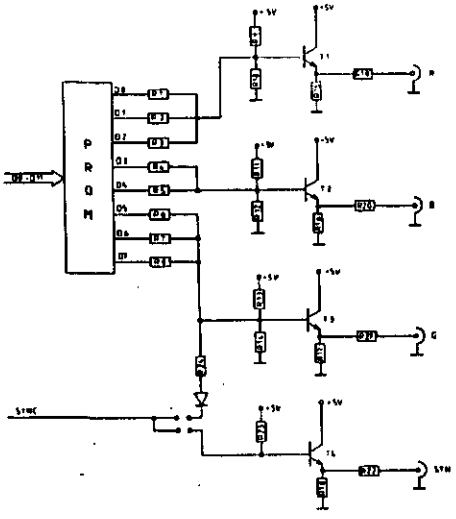
površini s srednjo prioriteto oziroma drugi ravnini naj pripadajo štirje biti in površini z najvišjo prioriteto oziroma tretji ravnini pet bitov. Iz tega sledi, da bi lahko v posameznih površinah realizirali naslednje število barv: 8 v prvi površini, 16 v drugi površini in 32 v tretji površini. Ker pa so površine razdeljene po prioriteti, mora biti ta nekje vpisana. Za to moramo žrtvovati eno barvo. V prvo površino ni potrebno vpisati tega podatka, v ostali dve pa ga moramo, zato lahko realiziramo v drugi površini 15 barv in v tretji površini 31 barv. Kode barv, ki jih vpišemo v PROM za vsako površino posebej, so lahko popolnoma različne za vsako površino, nimajo nobene medsebojne odvisnosti.

Sedaj pa poglejmo, kako to realiziramo v PROM-u. Na najnižjih osem naslovnih mest PROM-a vpišemo kode za barve prve površine. Sledi vpis kod za barve druge površine, kjer pa mora biti vsaka koda vpisana osem-krat. Najprej osemkrat vpišemo kodo prve barve in tako naprej, vse do petnajste barve. Sedaj sledi vpis kod za barve tretje površine. Te kode morajo biti vpisane na enak način kot za drugo površino, le da mora biti vsaka koda vpisana 16x8 krat, oziroma 128 krat. Če sedaj izračunamo, kakšen PROM rabimo za to LOOK UP tabelo, dobimo, da mora imeti 4K naslovnih mest. To število dobimo, če pomnožimo število vseh barv, ki bi jih lahko realizirali med sabo. To je 8x16x32. Isto vrednost pa dobimo, če izračunamo 2^n, kjer je n = 12 (n je število bitov iz slikovnega pomnilnika). Za navedeni primer rabimo zelo velik PROM, ki pa mora izpolnjevati tudi časovni pogoj. Biti mora zelo hiter.

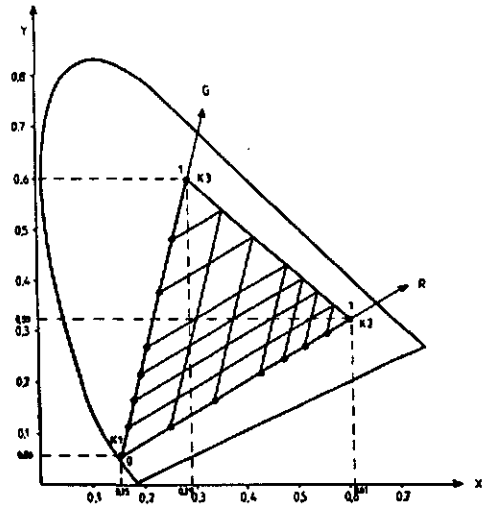
6. VEZJE ZA D/A PRETVORNIK

Sedaj poglejmo, kako smo v našem primeru rešili problem tvorjenja ustreznih napetostnih potencialov za R, G in B signale. Vzeli smo vezje na sliki 1. Ta slika prikazuje LOOK UP tabelo oziroma PROM, D/A pretvornik in vezje za tvorjenje sinhronizacije. Sinhronizacija je lahko samostojen signal, ki ga vodimo na barvni slikovni zaslon, ali pa je lahko skrita v signalu, ki predstavlja zeleno barvo. Upora R23 in R24 je potrebno izbrati tako, da so izpolnjeni pogoji, ki jih zahteva slikovni zaslon. Tranzistorji T1 do T4 so uporabljeni kot napetostni sledilniki. Upori R15 do R18 služijo za tokovno definicijo. Z upori R19 do R22 dosežemo uporovno zaključitev 75E, ki je potrebna za koaksialni kabel. Z upori R1 do R14 tvorimo ustrezne potenciale na bazah tranzistorjev in s tem napetosti na vhodih v slikovni zaslon.

Ker iz PROM-a dobimo osem bitov, barve pa so tri, ne moremo vsaki barvi dodeliti enako število bitov. Odločili smo se, da rdeči in zeleni barvi dodelimo po tri bite, modri barvi pa samo dva. Tako je mogoče doseči po osem napetostnih nivojev za rdečo in zeleno barvo, za modro barvo pa lahko realiziramo le štiri napetostne nivoje. S paralelno vezavo uporov R1 do R3 z R10 spreminjamo napetosti za rdečo barvo, z upori R4, R5 in R12 spreminjamo napetosti za modro barvo in z upori R6, R7, R8 in R14 spreminjamo napetosti za zeleno barvo.



Slika 1: Prikaz vezja za tvorjenje napetostnih nivojev



Slika 2: CIE diagram z vrisaim trikotnikom barvne zmogljivosti zaslona

Ta paralelna povezava je možna, ker ima PROM tako imenovane oppen collector izhode. Paralelno povezavo teh uporov izvedemo s postavljanjem izhodov PROM-a v nizek nivo. V odvisnosti od vrednosti teh uporov določimo, kakšno kombinacijo visokih in nizkih nivojev moramo nastaviti na izhodih PROM-a, da bodo napetostni nivoji na vhodih v slikovni zaslon ustrezali željeni barvi. Tako se ob postavitvi samih enk na vseh izhodih pojavi največja napetost, ki ustreza beli barvi. Ob postavitvi samih ničel pa se na izhodih pojavijo najnižje napetosti, ki ustrezajo črni barvi. S postavitvijo vmesnih stopenj pa dobimo še ostale barve.

7. IZRAČUN VREDNOSTI ELEMENTOV D/A PRETVORNIKA

V CIE diagram smo vnesli tri točke osnovnih barv slikovnega zaslona in s povezavo le-teh dobili trikotnik, ki predstavlja barvno zmogljivost zaslona. Izdelamo si nov koordinatni sistem, ki ima izhodišče v modri točki (točka K1 na sliki 2). Rdeča os gre od točke K1 proti točki K2 in zelena os proti točki K3. Vse te točke poda proizvajalec. Enote tega sistema so razvidne iz slike 2. Sedaj na rdečo in zeleno os vnesemo po osem točk. Skozi te točke povlečemo vzporednice z osema in dobimo vrsto presečišč. Ta presečišča približno predstavljajo barve, ki jih želimo na zaslonu. Napaka nastane zaradi tega, ker so osnovne barve že mešanica barv. Ostale barve, za katere ni presečišč, bodo imele enake barvne tone kot te, le da bodo manj svetle. Izbranim točkam določimo koordinate. V tem diagramu velja pravilo, da je vsota vseh treh koordinat enaka ena. Iz tega sledi, da lahko izračunamo vrednosti tudi za modro barvo. Ker pa v našem primeru lahko realiziramo le štiri različne vrednosti za modro barvo, izberemo izmed vseh vrednosti za modro barvo tiste štiri, ki najbolj ustrezajo želenim barvam. Vrednosti teh koordinat pretvorimo v napetostne potenciale. Za vsak barvni zaslon je podana maksimalna in minimalna napetost na vhodih R, G in B. Za naš primer je to 1,4V in 0,4 V. Napetost 1,4 V ustreza vrednosti 1 v koordinatnem sistemu, napetosti 0,4 V pa vrednost Ø. Vmesno področje se deli linearno. Na osnovi tega lahko za vsako odčitamo vrednost iz diagrama, torej za vsako barvo, določimo, kakšna mora biti izhodna napetost. Ker pa mi ne moremo spreminjati direktno izhodne napetosti, ampak jo lahko spreminjamo posredno, moramo napetost na bazi tranzistorja spreminjati v drugačnih mejah, kot je izhodna napetost. Na bazi tranzistorja, kjer mi dejansko spreminjamo napetost, moramo napetost spreminjati v mejah od 1,5V do 3,5V in sicer zato, ker moramo upoštevati 75 ohmsko zaključitev koaksialnega kabla in padec napetosti na tranzistorju. Iz tega sledi, da moramo, če zavzema koordinata barve vrednost Ø, nastaviti na bazi tranzistorja napetost 1,5 V. Za koordinato 1 nastavimo napetost 3,5 V, vmesne koordinate potekajo li-

nearno. Vse napetosti izračunamo po naslednji enačbi:

$U = U_{min} + (U_{max} - U_{min}) \cdot K$

U = napetost na bazi tranzistorja

U_{min} = napetost na bazi tranzistorja za vrednost koordinate Ø

U_{max} = napetost na bazi tranzistorja za vrednost koordinate 1

K = koordinata barvne točke iz diagrama na sliki 2

Na osnovi U_{min}, U_{max}, in dopustne tokovne obremenitve PROM-a, določimo vrednosti uporov R9, R10 in neke upornosti R, ki jo vežemo vzporedno k uporu R10. Vrednosti teh uporov izračunamo na sledeč način:

$R = U_{min}/I_p$

$R9 = 5.R/U_{min} - 5.R/U_{max}$

$R10 = R9 \cdot U_{max}/(5 - U_{max})$

I_p = dopustna tokovna obremenitev PROM-a

R = minimalna upornost, ki jo ustvarimo z vzporedno vezavo uporov R1, R2 in R3.

Upornost, ki jo moramo vezati vzporedno k R10, izračunamo na naslednji način:

$R = U.R9.R10/(5.R10 - U.R9 - U.R10)$

Napetost U ustreza že prej izračunani napetosti za barve. Na ta način lahko izračunamo vse upornosti, s katerimi dosežemo vse željene svetlosti posameznih barvnih točk. Za rdečo barvo smo na primer dobili osem različnih vrednosti uporov. Ker pa imamo na razpolago le tri upore, le-te izberemo tako, da vzporedna vezava dveh ali vseh treh uporov da vse ostale vrednosti, ki niso realizirane direktno z izbranimi upori. Vseh teh upornosti ni mogoče točno realizirati in zato tudi barve niso povsem take, kot smo jih v začetku izbrali. Izbira uporov je najbistvenejši faktor, ki vpliva na točnost izbranih barv. Zgoraj je podan izračun za rdečo barvo, za ostale barve je potrebno le ustrezno spremeniti oznake uporov.

Potem, ko smo določili vrednosti vseh uporov, se lahko lotimo sestavljanja LOOK UP tabele. Podatke, ki jih moramo vpisati v PROM, sestavimo na osnovi odčitanih koordinat iz CIE diagrama. Za vsako osnovno barvo določimo izhodno kombinacijo bitov, ki pomeni ustrezno svetlost te barve. Te bite sestavimo v pravilnem vrstnem redu, kot ga zahteva povezava PROM-a in uporov. Tako pridemo do kode, ki jo vpišemo v PROM.

8. ZAKLJUČEK

Za vsako željeno barvo, po zgoraj opisanem postopku, določimo tri koordinate v spremenjenem CIE diagramu. Iz teh koordinat izračunamo ustrezne napetosti, ki jih rabimo za realizacijo teh barv. Na osnovi teh napetosti določimo vrednosti uporov D/A pretvornika. Sledi izdelava LOOK UP tabele, ki preko D/A pretvornika nastavi ustrezne izhodne napetosti. Za prikaz določene barve mora biti v slikovni pomnilnik vpisana kombinacija bitov, ki pomeni naslov kode barve v LOOK UP tabeli. Podatke za slikovni pomnilnik pa izračunava ustrezna programska oprema.

LITERATURA

1. Lee Baldwin: "Color considerations", Byte--September 1984
2. Mark Dahmke: "Scion color system", Byte--Julij 1982
3. Kastelic Marko: Diplomsko delo
4. CIE standard
5. Barco industries: Colour graphic display

BARVNI GRAFIČNI TERMINAL TELEINFORMACIJSKEGA SISTEMA TI-30

B.GRILEC, M.KASTELIC, M. MARKELJ, P.PETERLIN, N.PANIĆ, B.GROŠELJ
ISKRA-AVTOMATIKA, Ljubljana, Jugoslavija

UDK: 681.326

POVZETEK - Članek prikazuje problematiko in način realizacije grafičnega barvnega terminala. Opisana so osnovna načela računalniške grafike ter aparaturna in programska oprema grafičnega barvnega terminala za interaktivno vodenje procesov v realnem času.

ABSTRACT - The article presents the problem and manner of realisation of the graphics colour terminal. It defines basic principles of computer graphics, hardware and software of graphic colour terminal for interactive process control in real time.

1. UVOD

Računalniška grafika predstavlja kombinacijo računalnika in grafičnega zaslona. Na osnovi informacije na zaslonu se operater odloča in posreduje ukaze računalniku. Grafični barvni zaslon se uporablja skoraj povsod tam, kjer se uporablja računalnik.

Grafični interaktivni terminal je naprava visokih zmogljivosti z veliko možnostjo uporabe v vseh sferah človekovega udejstvovanja in ustvarjanja. Njihova cena nenehno pada, večajo pa se zmogljivosti terminalov.

Računalniško grafiko lahko po načinih uporabe delimo na /1/:

- poslovno informatiko (Management information),
- uporabo v znanstvene namene (Sceintific graphics),
- upravljanje procesov (Command and control),
- procesiranje slik (Image processing),
- generiranje slik v realnem času (Real-time image generation),
- konstruiranje mehanskih in električnih podsklopov (Electrical and mechanical design).

Barvni grafični terminal teleinformacijskega siatema TI-30 sestavlja en ali več monitorjev, alfanumerična in funkcijska tastatura, sledilna krogla, svetlobno pero ali grafična tablica. Terminal je glavni sestavni del operaterjevega delovnega mesta in je samostojna enota sistema TI-30. S hitrimi serijskimi vodili je povezana z njegovo podatkovno bazo.

Terminal je namenjen predvsem za vodenje procesov v realnem času.

1.1 . Grafični barvni terminal

S padcem cene pomnilnih vezij ter z visokointegriranimi krmilniki so se pojavili zelo sposobni grafični terminali, ki že sami rešujejo dokaj zapletene probleme. Razvile so se tudi vhodno-izhodne enote. V začetku so obstajale samo alfanumerične in funkcijske tastature, kasneje pa se je pojavilo svetlobno pero. Zaradi pomanjkljivosti (težko ga je dalj časa držati pravokotno na zaslon) se je uveljavila grafična tablica, hkrati pa tudi sledilna krogla in takoimenovana miška.

Pri vodenju nekaterih procesov pa potrebujemo še druge naprave: sinoptično ploščo, merilne instrumente - zlasti registrirce, opozorilne in alarmne naprave, rezervna komandna mesta itd.

Vsekakor je najpomembnejši grafični barvni terminal, ki rešuje problematiko človek - stroj s sintaktičnega in semantičnega vidika. S sintaktičnega vidika je treba sestaviti take informacijske strukture, da ni presežena človekova sposobnost prevzema in obdelave informacij. S semantičnega vidika pa morajo biti sestavi taki, da še omogočajo hitro in zanesljivo odločanje.

Za starejše centre vodenja je značilno, da se na velikih sinoptičnih ploščah paralelno pri-

kazuje veliko število informacij. Informacije, ki jih operater v danem trenutku ne rabi, so motilne.

Raziskave so pokazale, da je paralelni prikaz boljši od serijskega, če se prikazujejo le "čiste" informacije (brez motilnih). S tem so doseženi krajši reakcijski časi. Paralelni in serijski prikaz pa sta si enakovredna pri prikazih, ki vsebujejo tudi motilne informacije.

1.2. Naprave za interaktivno delo operaterja s terminalom

1.2.1. Alfanumerična tastatura

Med klasične naprave prištevamo alfanumerično tastaturo. Današnje tastature imajo izpopolnjeno obliko tipk, tako da tipkanje ni utrudljivo. Krmiljene so z mikroprocesorjem in imajo v glavnem serijski izhod.

Za vodenje procesov sama alfanumerična tastatura ni primerna, ker operater vnaša kodirane podatke. S tem je velika možnost napak, zlasti v kočljivih situacijah, ko je potrebno hitro reagirati. Tako služi alfanumerična tastatura le za editiranje in pisanje komentarjev. To njeno pomanjkljivost odpravimo tako, da k alfanumerični tastaturi dodamo še funkcijsko tastaturo.

1.2.2. Funkcijska tastatura

Za razliko od alfanumerične tastature, ki je za računalnik le vhodna enota, je funkcijska tastatura vhodno-izhodna enota. V tipke funkcijske tastature so vgrajene LED diode, ki jih krmili mikroračunalnik. Stanja LED diod so naslednja:

- ne gori
- gori
- utripa

S funkcijsko tastaturo operater vodi proces /2/. Da tipko čim laže najde in da je čim manjša možnost napačne izbire, so polja tipk organizirana tako, da

- jih operater zajame z enim pogledom,
- tvorijo različne vzorce.

Funkcijska tastatura je razdeljena na naslednja polja:

- sistemske funkcije,
- izbira slik večjih področij procesa,
- izbira slik podpostaj, tabel, krivulj
- nastavljanje parametrov in izdaja komand.

Zaradi teh lastnosti ima funkcijska tastatura naslednje prednosti:

- majhna možnost napačnega posega,
- operater v kritičnih situacijah hitro najde ustrezno tipko,
- vrstni red posameznih operacij je programiran,
- operater se jo hitro nauči uporabljati

1.2.3. Sledilna krogla

Ker je slikovni pomnilnik večji od prikazanega, s sledilno kroglo pomikamo okno po pomnilniku. Ločimo dva tipa sledilnih krogel:

- sledilna krogla oddaja le impulze (x+, X-, -y+, y-), ki jih mora šteti določena enota in na podlagi le-teh mora mikroračunalnik izračunati relativni premik;
- sledilna krogla oddaja že preračunane premike.

Prednost slednje je v tem, da mikroračunalnik ne potroši časa pri preračunavanju števila impulzov, ter da lahko mikroračunalnik med delovanjem menjava režim delovanja sledilne krogle.

Pomik okna lahko simuliramo tudi s pomočjo uporabe večih tipk na alfanumerični tastaturi, vendar pa je to bolj zapleteno, saj mora operater za določeno smer izmenično pritiskati več tipk.

Med editiranjem slik lahko sledilno kroglo uporabimo za premik kurzorja.

Namesto sledilne krogle se lahko uporabi tudi miška, čeprav ima za vodenje procesa sledilna krogla prednosti.

1.2.4. Svetlobno pero

Svetlobno pero se uporablja za izbiro menija, spreminjanja statusa določenih polj ali pa samo znakov. Z njim lahko pošiljamo komande, skratka prevzema funkcije alfanumerične funkcijske tastature.

Slabosti pri uporabi svetlobnega peresa so zlasti naslednje:

- operater mora držati pero pravokotno na zaslon in se pri tem utrudi;
- v kritičnih situacijah je možnost napak večja;
- zaradi motilnih vplivov (drugi izvori svetlobe) se lahko prečitajo nepravilni podatki.

Zaradi navedenega se vedno pogosteje uporabljajo grafične tablice.

1.2.5. Grafična tablica

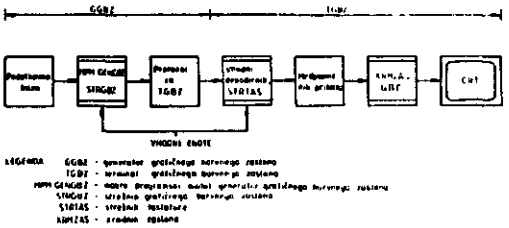
Uporaba grafične tablice je veliko lažja od uporabe svetlobnega peresa. S folijami, ki jih položimo na tablico, določimo funkcije posameznih površin, z dotikom peresa pa aktiviramo določen program. Površina tablice

je razdeljena na "tipke" in aktivno površino. Površina slednje ima različno funkcijo pri različnih aplikacijah.

2. ZGRADBA GRAFIČNEGA BARVNEGA TERMINALA TELEINFORMACIJSKEGA SISTEMA TI-30

Terminal je sestavljen iz dveh mikroračunalniških podsistemov TGBZ - terminala grafičnega barvnega zaslona in GGBZ - generatorja prikazov grafičnega barvnega zaslona.

Funkcijski model terminala prikazuje slika 2.1.

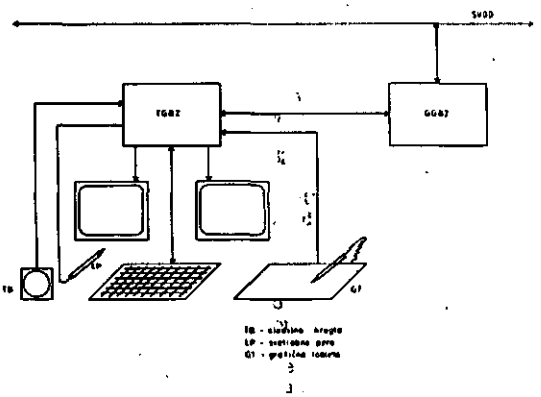


Sl.2.1: Funkcijski model interaktivnega grafičnega terminala sistema TI-30

Opomba: V obrobljenih kvadratih so procesorji, v praznih datoteke

TGBZ je krmilna enota, na katero so priključeni barvni monitor, alfanumerična tastatura, funkcijska tastatura ali pa še sledilna krogla, svetlobno pero ali grafična tablica.

GGBZ arhivira prikaze na diskovni enoti in generira prikaze na zaslonu. Nanj sta priključeni dve diskovni enoti (floppy in winchester). Oba sistema sta med seboj povezana s serijskimi ali paralelnimi vodili, slika 2.2.



Sl.2.2: Blokovna shema inteligentnega barvnega grafičnega terminala sistema TI-30

Terminal TGBZ omogoča interaktivno delo operaterja s sistemom pri vodenju procesov v realnem času.

Programski sklop KGBZ omogoče interakcijo operaterja z vodenim procesom preko monitorja in tastature. Opravlja naslednje funkcije:

- prikazuje alfanumerične in grafične znake različnega statusa,
- sprejema sporočila nadrejenega mikroračunalnika,
- pošilja sporočila v programski modul STRGBZ (preko programskega modula DPOV),
- sprejema operaterjeve ukaze s tastature in prižiga LED diode na funkcijski tastaturi
- ugotavlja pravilno zaporedje operaterjevih postopkov in sestavlja izhodna sporočila,
- označuje elemente na zaslonu s kurzorjem,
- sestavlja skelet novih prikazov.

2.1. Programski modul STRGBZ

Programski modul sprejema sporočila iz terminala TGBZ in opravlja naslednje funkcije /4/:

- sprejema sporočila iz PSKGBZ,
- kliče slike z diska,
- vpisuje slike na disk,
- briše datoteke na disku,
- prenaša slike v medpomnilnik na izpis,
- vpisuje opozorila na zaslon,
- pošilja sporočila v podatkovno bazo.

2.2. Makro programski modul GENGBZ

Sprejema dinamične podatke iz podatkovne baze in jih prikazuje na terminalu.

Opravlja naslednje funkcije:

1. Sprejema podatke iz distribuirane podatkovne baze multimikroračunalniškega sistema TI-30 preko dveh logičnih kanalov (za prikaz na dveh zaslonih).
2. S pomočjo tabel te podatke prepozna in nato v tabelah in v subrutinah poišče ustrezne podatke za prikaz.
3. Nove podatke preda v terminal GBZ.

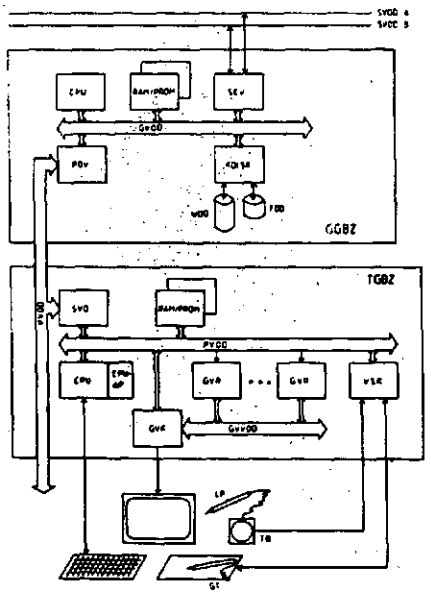
2.3. Aparaturna oprema

Terminal je zgrajen iz nabora modulov teleinformacijskega sistema TI-30. Slika 2.3. prikazuje zgradbo terminala.

V GGBZ so naslednji moduli:

CPU - procesor
RAM/PROM- delovno-programski pomnilnik
SEV - krmilnik serijskih vodil
KDISK - krmilnik diska
POV - povezovalnik vodil

Module povezuje GVOD - glavno vodilo. Povezava med GGBZ in TGBZ poteka po vertikalnem vodilu - VVOD.



Sl. 2.3: Zgradba inteligentnega grafičnega barvnega terminala

V TGBZ pa so naslednji moduli:

- CPU - procesor
- CPU-AP - aritmetični procesor
- SVO - spojnik vodil
- RAM/PROM - delovni pomnilnik
- GVK - grafični video krmilnik
- GVR - grafični video pomnilnik
- VSK - vmesnik za sledilno kroglo

Module povezuje LVOD - lokalno vodilo; modul GVK pa je z moduli GVR povezan z GVVOD-om - - grafično video vodilom.

Lastnosti modulov GVK in GVR ter VSK so podrobneje opisane v nadaljevanju.

3. OPIS APARATURNIH MODULOV RAČUNALNIŠKE GRAFIKE SISTEMA TI-30

3.1. Opis modula GVK

Modul GVK krmili delovanje barvnega monitorja. Tvori vse potrebne signale za njegovo delovanje, hkrati pa krmili tudi prenos podatkov med mikroprocesorskim modulom sistema TI-30 in slikovnim pomnilnikom GVR.

Modul je sestavljen iz naslednjih funkcijskih enot:

- krmilnik grafičnega monitorja - GDC krmilnik: krmili grafični monitor, skrbi za vpis in čitanje slikovnega pomnilnika, osvežuje slikovni pomnilnik, skrbi za zvezen pomik, povečavo slike ter ima vhod za svetlobni svinčnik LP;
- DMA krmilnik nadzoruje hitri prenos podatkov med pomnilnikom in GDC krmilnikom oziroma slikovnim pomnilnikom;
- arbitražno kontrolno vezje: krmili dodeljevanje vodila;
- časovno vezje: tvori časovne signale za GDC krmilnik, DACK vezje ter za vpis v slikovni pomnilnik;
- krmilni register: krmili posamezna vezja modula;
- register površine - SR: vklaplja posamezne površine;
- register zveznega pomika in povečave - ZPR: krmili zvezni pomik in povečavo slike na monitorju;
- DACK vezje: prilagaja signale za DMA prenos med DMA in GDC krmilnikom;
- krmilno vezje: izbira in krmili pomnilnik;
- "LOOK UP" tabela: pretvarja digitalno vrednost za določitev barve znaka na monitorju, D/A pretvornik pa to vrednost pretvori v analogni signal;
- dekodirnik naslovov: tvori potrebne naslovne signale;
- register pomnilne strani - PSR: naslavlja nastavljeno pomnilno stran, kó prevzame vodilo DMA krmilnik.

3.2. Opis modula GVR (grafični video pomnilnik)

Modul GVR služi za shranjevanje video informacij. Krmili ga modul GVK, ki osvežuje vsebino pomnilnih enot modula GVR, jo na zahtevo mikroprocesorja čita in vpisuje nove podatke v te pomnilne enote.

Slikovni pomnilnik obsega tri ravnine (3 biti/ /točko).

Na modul GVK so lahko priključeni do štirje moduli GVR.

Vsebina slikovnega pomnilnika se preko pomikalnih registrov prenese na modul GVK in od tu na monitor.

Modul je sestavljen iz naslednjih funkcijskih enot:

- slikovni pomnilnik (RAVNINA A, RAVNINA B, RAVNINA C):ravnine so organizirane 16-bitno (vsaka ima 16 pomnilnikov DRAM 64K x 1 oz. 256 x 1);
- krmilno vezje: krmili vpisovanje, branje in osveževanje slikovnega pomnilnika;
- ojačevalnik adres: ojačuje adresne signale;
- podatkovni ojačevalnik : krmili podatkovne signale, ko GDC krmilnik modula GVK čita posamezno ravnino;
- pomikalno in zakasnilno vezje: sprejeto pa-

ralelno 16-bitno besedo pretvori v serijsko obliko in informacije posreduje modulu GVK.

3.2.1. Organizacija pomnilnika

Raznolikost uporabe računalniške grafike je povzročila, da sta se razvili dve področji, dva načina organizacije pomnilnika:

- bitna organizacija
- znakovna organizacija.

Pri bitno organiziranem slikovnem pomnilniku je pomnilnik organiziran kot matrika naslovljenih točk. Za vsako točko na zaslonu je v pomnilniku shranjena vrednost intenzitete in barve. Vsaka točka je lahko vključena ali izključena. Z enostavnimi ukazi lahko vsebino pomnilnih celic komplementiramo, setiramo, resetiramo, ji spremenimo atribut itd. Pri taki organizaciji pomnilnika lahko enostavno vnašamo podatke v pomnilnik z uporabo svetlobnega svinčnika ali grafične tablice, miške ali druge vhodne enote.

Znakovno organiziran slikovni pomnilnik ima pomnilnik organiziran kot zaporedje kod. Vsak znak ima svojo kodo, ki predstavlja na zaslonu matriko točk (9 x 12). Za vsak znak so v pomnilniku shranjeni še atributi (utripanje, barva znaka, barva ozadja, intenzivnost, podčrtavanje,...).

Vsekakor potrebuje znakovna organizacija malo slikovnega pomnilnika, po drugi strani pa so tu omejene grafične sposobnosti terminala. Zaradi majhnega pomnilnika se uporabljajo le statična RAM vezja. Vedno večja kapaciteta in nizka cena dinamičnih pomnilnikov pa omogočata gradnjo bitno organiziranih grafičnih terminalov velikih zmogljivosti.

3.2.2. Ločljivost in format slike

Pri zasnovi grafičnega terminala moramo najprej izbrati ločljivost in hitrost skaniranja. Vsi konvencionalni rastrski monitorji imajo zaslon (širina/višina) v razmerju 4:3/5/. To razmerje se imenuje aspect ratio.

Če želimo imeti kvadratno točko na zaslonu (pixel), potem mora biti število horizontalnih točk 4/3 števila vertikalnih točk, kar se označuje kot 4H - 3V. H in V predstavljata število točk v horizontalni in vertikalni smeri. Resolucija je odvisna od celotnega števila točk, ki mora biti potenca števila 2. Če ni, mora biti število točk zaokroženo na naslednjo višjo potenco števila 2.

Zahteva je še večja. V horizontalni smeri mora biti število točk sodi mnogokratnik števila 16. Za prikaz 512H x 512V = 2exp(18)=262144 točk. Format 576H x 432V izpolnjuje vse pogoje. Število vseh točk je tako 248832 in razmerje horizontalnih z vertikalnimi točkami je 4/3. Število horizontalnih točk je točno 36 16-bitnih besed.

Ker pa moramo imeti za barvno grafiko najmanj tri bite na točko, potrebujemo 3 x večji slikovni pomnilnik. Velikost pomnilnika je odvisna tudi od načina naslavljanja GDC krmilnika.

Skaniranje slike mora biti hitrejše od občutljivosti očesa, da preprečimo migotanje slike. Torej je za eno sliko na razpolago 20 ms pri 50 slikah/sekundo. Pri animacijah se ta čas zmanjša, saj je priporočljivo, da je do 68 slik na sekundo oziroma je čas 1 slike 14,7 ms. Aktivni čas prikazovanja je približno 75 % celotnega časa, zatemnitev pa traja 25% časa. Razmerje med aktivnim časom in časom zatemnitve je odvisno od monitorja in GDC krmilnika. Pri boljših monitorjih dobimo slabše rezultate predvsem zaradi dolge horizontalne zatemnitve, ki jo potrebuje GDC krmilnik.

Pri skaniranju slike so istočasno aktivne vse tri ravnine - GDC krmilnik naslovi isti naslov v vseh treh ravninah, podatki pa se pretvorijo v serijsko obliko in pošljejo na monitor.

Slikovni pomnilnik je organiziran trikrat po 16K bitov 16-bitnih besed.

Pri risanju grafičnih figur se slikovni pomnilnik obravnava kot velika ploskev, ki je razdeljena na tri osnovne barve: rdečo, zeleno in modro. Tako lahko spodnji del slikovnega pomnilnika predstavlja rdečo barvo, srednji del zeleno in višji modro barvo. Za vsako točko v osnovni barvi mora biti v slikovnem pomnilniku setiran en bit. Kadar želimo imeti štiri barve, sta za vsako točko potrebna dva bita v slikovnem pomnilniku, hkrati pa porabimo za vpis teh dveh bitov dva cikla in tako se sistem upočasni. Za večje število barv potrebujemo sorazmerno večje število vpisnih ciklov.

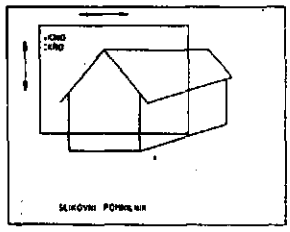
Ker so danes dostopni pomnilniki večjih kapacitet (64K bitov), je lahko format slike večji od 512H x 512V. Zelo se je uveljavil format 640H x 480V. Pri skaniranju slike s frekvenco 50 Hz je za eno sliko na razpolago 20ms. Ker potrebuje monitor za vertikalno zatemnitev 1,25 ms, ostane za skaniranje 480 vrstic aktivnih še 18,75 ms.. Z deljenjem 18,75 ms s 480 vrsticami dobimo 39,062 µS/vrstico oziroma ekvivalentno horizontalno frekvenco 25,6 KHz. Horizontalna zatemnitev potrebuje 7 µS /vrstico, aktivni del vrstice pa je 32,062 µS. V tem času je prikazanih 640 točk oziroma za eno točko po-

trebujemo 32.062 uS/640 točk, kar znaša 50.096 ns. To ustreza točkovni frekvenci 19,96 MHz.

3.2.3. Konfiguriranje terminala z moduli GVR

Eden najvažnejših delov rastrske grafike je slikovni pomnilnik, v katerem je shranjena prikazana slika na zaslonu.

Slikovni pomnilnik je večji od prikazane slike na zaslonu. Mikroprocesor lahko premika okno po slikovnem pomnilniku zvezno v horizontalni ali vertikalni smeri.

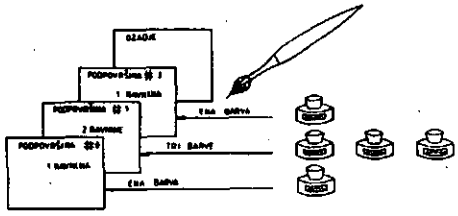


Slika 3.1: Pomikanje okna po slikovnem pomnilniku

Če želimo menjati sliko, medtem ko žarek otipava zaslon, dobimo na zaslonu motnjo. Da to preprečimo, lahko vpisujemo nove podatke v slikovni pomnilnik le ob vertikalni zatemnitvi. Kadar imamo kritične časovne razmere, lahko manjše spremembe vpisujemo kadarkoli, ker motnja ni tako huda, da bi motila operaterja.

Slikovni pomnilnik modula GVR je organiziran kot površina, ki je sestavljena iz ravnin. Površina je slikovni pomnilnik, ki je ekvivalenten modulu GVR in ima vedno 3 bite/ /točko oziroma površino sestavljajo tri ravnine.

Ravnine pa lahko grupiramo v podpovršine. Število podpovršin je odvisno od števila ravnin, ki jih vsebuje podpovršina.



Slika 3.2.: Število barv v odvisnosti od števila ravnin podpovršine

Število barv - govorimo lahko o barvah črnila-je odvisno od števila ravnin, ki jih zajema ena podpovršina. Podpovršina z eno ravnino ima eno barvo, podpovršina z dvema ravninama ima tri barve, podpovršina s tremi ravninami ima sedem in podpovršina s štirimi ravninami petnajst barv /6/.

Površina
Velika odlika modulov GVR je dejstvo, da lahko slikovni pomnilnik organiziramo v različnih velikostih in oblikah. To nam omogoča koncept površin.

Površino sestavlja eden ali več modulov GVR. V sistemu imamo lahko največ do štiri površine.

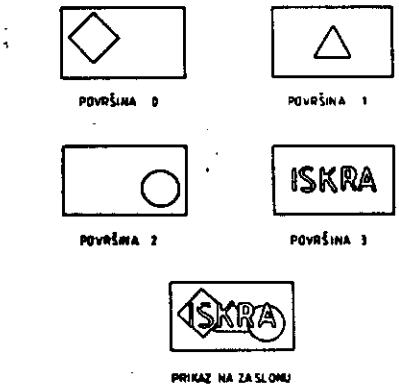
Velikost površine definira število modulov GVR, obliko površine pa definiramo s PITCH parametrom pri organizaciji GDC krmilnika.

Če uporabljamo več površin lahko prikažemo vsako posebej kot posebno stran ali pa tvorimo razne funkcije med površinami. Sestavljen prikaz lahko generiramo s prekrivanjem ravnine na ravnino po vnaprej določeni prioriteti, lahko vklapljamo posamezne površine ali pa imamo kombinacijo obeh možnosti oziroma s kreiranjem LOOK UP tabele tvorimo različne funkcije med podpovršinami. Slika 3.3 prikazuje, kako so lahko kombinirane površine pri uporabi tehnike prekrivanja. Površine imajo različno prioriteto. Najvišjo prioriteto ima najvišja površina. Tako trikotnik prekriva kvadrat, krog prekriva kvadrat in trikotnik, napis ISKRA prekriva vse. Vsaka površina lahko postane nevidna z vpisom v register površine na modulu GVK.

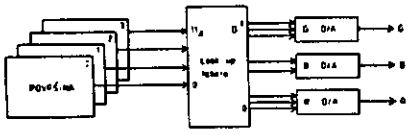
Če uporabljamo več površin, lahko uporabljamo do 8 ravnin za različne barve, ostale (do 12) pa za prekrivanje, za počasno ali hitro utripanje ali pa za kurzor.

Vezje, ki ga prikazuje slika 3.4., prikazuje uporabo LOOK UP tabele, ki je med slikovnim pomnilnikom in D/A pretvornikom.

Vsaka površina ima tri ravnine, zato ima LOOK UP tabela 12 vhodnih adresnih linij in 8 izhodnih podatkovnih linij.



Sl. 3.3: Prekrivanje površin



Sl. 3.4: Uporaba LOOK UP tabele

Modul VSK je sestavljen iz naslednjih funkcijskih enot:

- enosmerni ojačevalnik ojačuje signale, ki jih modul dobiva z mikroračunalniškega vodila oz. jih modul oddaja na vodilo;
- dvosmerni ojačevalniki: ojačujejo podatkovne signale;
- adresni dekoder z nastavljalnikom adres: dekodira adresne signale in jih primerja z nastavljenimi, aktivira modul;
- števci štejejo impulze sledilne krogle, ki jih dobijo od krmilnega vezja; dajejo zahtevo za programsko prekinitev;
- krmilno in prilagoditveno vezje: preoblikuje signale sledilne krogle;
- prilagoditveno vezje: sestavljeno je iz linijskega sprejemnika, ki pretvarja simetričen signal LPI v asimetrično obliko, ki jo potrebuje modul GVK.

4. ZAKLJUČEK

Terminal je modularno grajen tako, da ga za različne aplikacije dopolnjujemo z različnimi moduli.
Tako imamo lahko večje število monitorjev v TGBZ-ju ali pa do pet TGBZ-jev na GGBZ-ju.

Terminal je realiziran tako, da prikazuje ASCII in posebne grafične znake kakor tudi poljubne krivulje.

Omogoča:
- zvezni horizontalni pomik slike;
- prekrivanje površine s površino ali ravnino;
- istočasno lahko prikaže do 256 barv;
- vpisovanje in branje slikovnega pomnilnika lahko poteka s pomočjo mikroprocesorja ali DMA krmilnika;
- omogoča dve hitrosti utripanja;
- prikaz kurzorja v posebni ravnini.

Prav tako pa GDC krmilnik lahko sam riše:
- točke, linije, loke, pravokotnike in poljubne grafične ali ASCII znake znotraj slikovnega pomnilnika velikosti 1K krat 1K točk;
- krmili monitor resolucije 640x480 ali 1024x768 točk;
- omogoča povečavo slike do 16 x;
- v pomnilnik lahko vpiše do 16x povečan znak;
- na zaslonu sta lahko istočasno dve sliki, ki se lahko neodvisno premikata.

Na terminal se lahko priključi:
- alfanumerična tastatura;
- funkcijska tastatura;
- sledilna krogla;
- svetlobno pero;
- grafična tablica.

LITERATURA

/1/ C.Machover: Graphic displays, IEEE Spectrum 14 No.8, No.10. 1977
/2/ Peter Peterlin: Magistrsko delo, Fakulteta za elektrotehniko v Ljubljani, 12,1980
/3/ P.Peterlin,N.Panić,B.Grilec,M.Markelj: Sistem človek naprava v centrih daljinskega vodenja železniškega prometa, JUREMA, Zagreb, 1985
/4/ ISKRA-AVTOMATIKA,Razvojni inštitut: Splošni opisi programskih modulov, 1985
/5/ G.Palma,M.Olson,R.Jollis: Graphics display Controller, Electronic Design, 1, 83
/6/ R.Linsalata, R.Scalea: Raster graphics: expanding its frontiers, Computer Design, 7, 1981
/7/ B.Grilec,Magistrsko delo, Fakulteta za elektrotehniko v Ljubljani, 6, 1985

NAMESTITVENI ALGORITEM

Dušan PAGON, Institut "Jožef Stefan" Ljubljana

UDK: 519.174

Eden od problemov, s katerimi se srečujemo pri izdelavi tiskanih vezij, je optimalna namestitev komponent vezja v ravnini. Prispevek je posvečen problemu avtomatičnega nameščanja komponent pri načrtovanju elektronskih vezij. Module prirejamo pozicijam na plošči s ciljem minimizirati skupno dolžino vseh bodočih povezav med moduli (v smislu Manhattan razdalje). Za reševanje tega problema smo uporabili računsko ugoden algoritem linearnega prirejanja. Vse module smatramo za pravokotnike, pri čemer fiksiramo eno od njihovih dimenzij. Algoritem nameša komponente vzporedno eno z drugo (glede na izbrano fiksno dimenzijo), da bi olajšal bodoče razpeljevanje povezav.

AN ALGORITHM FOR THE AUTOMATIC PLACEMENT OF CIRCUIT MODULES

The two-dimensional placement is one of the problems which we meet while constructing a printed board. In the paper we deal with the problem of automated placement of electronic components in a circuit layout. The modules are assigned to the grids on the printed-circuit board so as to minimize the total wire lenght (relatively to the Manhattan distance). For solving this problem the computationally powerful linear assignment algorithm is used. All the modules are assumed to be rectangular and one of their dimensions is fixed. The algorithm provides a parallel placement of the components (relatively to the choosen fixed dimension) so as simplify the further routing.

UVOD

Pri konstrukciji elektronskih vezij naletimo na problem nameščanja komponent. Na pravokotno področje je treba namestiti množico komponent (integriranih vezij, diskretnih elementov, konektorjev,...), na katerih se nahajajo na določen način razporejeni priključki (elektične sponke). V logični shemi vezja so podane množice priključkov, ki morajo biti v vezju med seboj elektično povezani - to so tako imenovane signalne množice ali signalna omrežja. Te množice morajo biti disjunktne, ne smejo, torej, imeti skupnih priključkov. Namestitveni problem (the placement problem) je tedaj v tem, da namestimo dane komponente, upoštevajoč omejitve, tako, da bo dosežen minimum skupne dolžine stehtanih povezav med komponentami, ali pa se bomo temu minimumu vsaj dovolj približali. Pri nameščanju imamo običajno še dodatne sistematične omejitve za pozicije in smeri nameščanja komponent, povezane z gostoto povezav na plošči in legami priključkov na modulih. Pravilna namestitev mora poleg tega, da zadovoljuje najrazličnejšim tehničnim zahtevam, zagotoviti enostavnost in preglednost vezja, ki sta bistvenega pomena za zmanjšanje stroškov njegove izdelave in vzdrževanja.

Pri ukvarjanju z namestitvenim problemom se je treba zavedati, da so že zelo preproste verzije te naloge (npr. enodimenzionalno nameščanje) dokazano NP-polni problemi, kar pomeni, da se nima smisla gnati za ekzaktnimi rešitvami, ampak se je treba opreti na hevristične postopke, ki nam sicer dajo samo približno rešitev, vendar ob sprejemljivi porabi prostora in časa.

ALGORITEM LINEARNEGA NAMEŠČANJA

Predlagani postopek za nameščanje temelji na uporabi znanega algoritma za linearno prirejanje določenega števila objektov enakemu ali večjemu številu prostih pozicij. Pri tem je podana matrika A , element a(i,j) katere je ocena ustreznosti i-te pozicije j-temu objektu, naša naloga pa je najti minimum vsote elementov a(i,P(i)) po vseh permutacijah P . Možnost uporabe algoritma linearnega prirejanja pri reševanju namestitvenega problema je prvi omenil Akers [1]) v zvezi z nameščanjem enako velikih kvadratnih modulov.

Formalizacija, s pomočjo katere smo mi reducirali problem kvadratičnega prirejanja, kakršen je naloga o nameščanju, na linearno prirejanje, je naslednja. Predpostavljamo, da imajo komponente obliko pravokotnikov s širino 1 in poljubno celoštevilčno dolžino (ki ne presega dimenzij plošče). Podobna omejitev je smiselna in jo srečamo pri večini algoritmov za reševanje namestitvenega problema , glej recimo [2] , kjer je opisan postopek, ki ga uporabljajo v laboratorijih SHARPa. Na plošči uvedemo pravokotne koordinate in načrtamo mrežo s korakom 1. Module bomo nameščali v vozlišča te mreže izključno v vodoravni smeri, da bi omogočili najboljšo možno prehodnost dane strani plošče v tej smeri. Naj Term(k) pomeni množico priključkov komponente k . Če so koordinate delov komponent k in l v izbrani mreži enake (x(k),y(k)) in (x(l),y(l)) , ustrezno, potem bomo dolžino povezave med priključkoma p iz Term(k) in r iz Term(l) izračunali po formuli za Manhattan razdaljo:

$$d(p,r) = |x(k) - x(l)| + |y(k) - y(l)|.$$

Stehtana dolžina povezave med dvema moduloma bomo rekli vsoti dolžin povezav med vsemi njunimi priključki, pri čemer določene signale glede na njihovo pomembnost lahko štejemo večkrat (veliko težo je treba, naprimer, pripisati povezavi med sosednjima deloma iste komponente večje dolžine, ker bi sicer lahko prišli do nesmiselne namestitve).

KONSTRUIRANJE KVAZIOPTIMALNE NAMESTITVE

V sredino plošče namestimo modul z največjo vsoto stehtanih povezav. Na vsakem koraku na eno od prostih pozicij, sosednjo z že zasedenimi, namestimo tak modul, ki bo najmanj prispeval k skupni dolžini povezav. (V tem je razlika z že omenjenim algoritmom, ki ga uporabljajo pri SHARPu, ker slednji najprej vse komponente razporedi v vrstice, nato pa prične z urejanjem vrstic.) Ta modul določimo na naslednji način. Naj bo P prosta pozicija, k pa komponenta, ki zanjo kandidira. Razvrstimo razdalje d(P,R) pozicije P do vseh še prostih pozicij R v naraščajoče zaporedje:

$$d(P,R1) \ , \ d(P,R2) \ , \ \ldots \ , \ d(P,Rn) \ ,$$

vrednosti stehtanih povezav med komponento K in vsemi še nenameščenimi komponentami L pa v padajoče zaporedje:

$$c(K,L1) \ , \ c(K,L2) \ , \ \ldots \ , \ c(K,Ln) \ .$$

Vsoti produktov $1/2.d(P,Ri).c(K,Li)$, $i=1,2,\ldots,n$ dodamo vse produkte oblike $d(P,Qt).c(K,t)$, kjer t preteče vse že nameščene komponente, Qt pa je pozicija, ki jo zaseda komponenta t . Dobljeno število označimo z a(P,K) in nam bo služilo za oceno spodnje meje skupne dolžine povezav nastalih z namestitvijo modula K na pozicijo P . Matriko sestavljeno iz elementov a(P,K) obdelamo z algoritmom za linearno prirejanje. V ta namen uporabljamo varianto, ki sta jo predlagala Dinic in Kronrod v [3]. Pri njunem postopku se v vsaki vrstici izbere minimalni element, tako imenovani temelj, nato pa se določeni od temeljev zamenjujejo z novimi, dokler ni po eden prisoten v vsakem stolpcu naše matrike.

Na vsakem koraku namestimo tisti modul, ki naj bi glede na rešitev problema linearnega prirejanja zasedel najnižjo pozicijo, z ostalimi pa postopek ponovimo. Komponento večje dolžine v celoti namestimo na tistem koraku, ko je določen za namestitev prvi od njenih delov. Algoritem v bistvu na vsakem koraku predlaga namestitev vseh komponent, ki jo zatem iterativno izboljšuje.

S ciljem maksimalizirati ploščino območja, na katero se nameščajo moduli, pri njegovem omejenem obsegu, smo izbrali zanj obliko pravokotnika s skoraj enakimi stranicami. Daljšo od obeh stranic pravokotnika, s katerima se pričnejo iteracije, algoritem po potrebi še rahlo podaljša, tako da izdela namestitev vedno, kadar je vsota ploščin modulov, ki jih je treba namestiti, manjša od ploščine območja, na katero jih nameščamo.

ZAKLJUČEK

Čas, potreben našemu algoritmu za namestitev N komponent (komponente večje dolžine so štete z ustreznimi faktorji) je na računalniku VAX-750 z operacijskim sistemom VMS približno enak $10^{-5}.N^4$, kar pomeni, da je algoritem nekoliko hitrejši od tistih, ki uporabljajo bolj zapletene matematične konstrukcije(npr. [4]), po kvaliteti izdelane namestitve pa ne zaostaja bistveno za njimi.

Algoritem dopušča, da vnaprej definiramo določena "prepovedana" območja na plošči, v katera ne bo nameščal modulov, širino kanalov med komponentami, potrebnih za razpeljevanje povezav in napajanj, pa določimo tako, da ustrezno povečamo dimenzije vseh modulov.

Pomanjkljivost algoritma je zaenkrat v ireverzibilnosti osnovnega procesa, kar pomeni, da program ne zna sam delno podreti izdelane namestitve in jo dopolniti na drugačen način.

Na priloženi sliki je rezultat namestitve osemnajstih različno velikih pravokotnih modulov, matrika stehtanih povezav med katerimi je podana z $a(i,j) \equiv i+j \pmod{10}$.

LITERATURA:

1. S.B. Akers, ON THE USE OF LINEAR ASSIGNMENT ALGORITHM IN MODULE PLACEMENT. Proceedings of the 18th Design Automation Conference, 1981.

2. T. Kombe, T. Chiba, S. Kimura, T. Inufushi, N. Okuda, I. Nishioka, A PLACEMENT ALGORITHM FOR POLYCELL LSI AND ITS EVALUATION. Proceedings of the 19th Design Automation Conference, 1982.

3. E.A. Dinic, M.A. Kronrod, ODIN ALGORITM RESENIJA ZADACI O NAZNACENII. Doklady Akademii nauk SSSR. Tom 189, No.1, 1969.

4. K. Fukunaga, S. Yamada, H.S. Stone, T. Kasai, PLACEMENT OF CIRCUIT MODULES USING A GRAPH SPACE APPROACH. Proceedings of the 20th Design Automation Conference, 1982.

1	13	16	14 *1	14 *2	19				
7 *1	7 *2	4 *2	4 *1	2 *1	2 *2	9			
6 *2	6 *1	1 *1	1 *2	1 *3	10 *2	10 *1			
17	12 *2	12 *1	15	3 *3	3 *2	3 *1			
18	11	8 *1	8 *2	5 *3	5 *2	5 *1			

Vsota stehtanih povezav: 8556.0

KONCEPCIJA CAD SISTEMA ZA PRORAČUN ZAGRIJANJA TRANSFORMATORA U PROTUEKSPLOZIVNOJ IZVEDBI

Davor Zvizdić,
Elektrotehnički institut "Rade Končar"
Zagreb

UDK: 681.3.014

SAŽETAK: Opisan je CAD sistem za proračun zagrijanja transformatora u protueksplozivnoj izvedbi, koji omogućuje projektantima da odaberu povoljnu konfiguraciju transformatora s obzirom na zagrijanje u vrlo kratkom vremenu. Matematički model za proračun baziran je na mrežnoj analogiji. Stacionarna stanja se opisuju sistemom nelinearnih algebarskih jednadžbi. Vremenski tok promjene temperatura opisan je sistemom nelinearnih diferencijalnih jednadžbi.
Za predikaciju koeficijenata prijelaza topline u prirodnom konvekcijom hlađenim vertikalnim kanalima transformatora koristi se dodatni dvodimenzionalni proračun polja brzina i temperatura. Interaktivni programi omogućuju selektivni grafički prikaz dobivenih rezultata.

CONCEPT OF A CAD SYSTEM FOR CALCULATION OF HEAT TRANSFER IN FLAMEPROOF MINING TRANSFORMERS - This paper presents a CAD system for calculation of heat transfer in flameproof mining transformers, which enobles designers to develop satisfactory configuration, regarding temperature rise, in a very short time. The mathematical model used for calculation is based on network analogy. Steady state is described by a set of nonlinear algebraic equations. The transient response is described by a set of nonlinear differential equations. Additional, two-dimensional temperature and flow field calculation is used for prediction of heat transfer coefficients in natural convection cooled vertical chanels of the transformer. Interactive programs enable selective graphical presentation of calculated data.

1. UVOD

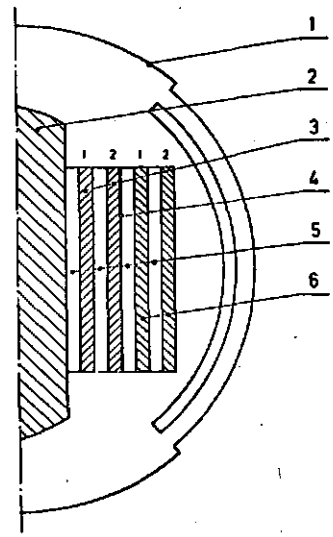
Složeni uređaji se sastoje od komponenata koje su izrađene od različitih materijala i koje imaju različite geometrije i funkcije. Tokom njihovog rada oslobađa se toplina koju je potrebno odvoditi. U strukturalnim sistemima odvijaju se različiti procesi simultano i njihovi elementi mogu, pored toga što su u toplinskoj, biti u hidrodinamičkoj, električnoj ili drugoj interakciji. Model koji služi za opis simultanih procesa može se izgraditi od modela individualnih, "bazičnih", procesa koji ga sačinjavaju. Ima raznih koncepata modela koji se mogu koristiti u ovu svrhu. U svakom slučaju pristup treba zadovoljiti slijedeće zahtjeve:

- jasno ocrtava fizikalnu bit procesa koji se modelira
- omogućuje povećanje točnosti rafinacijom modela
- može se jednostavno provjeriti
- pogodan je za CAD
- omogućuje nadogradnju i proširenje
- ne postavlja prevelike zahtjeve na memoriju računala.

Za modeliranje prijelaza topline u uređajima koji se sastoje od mnogo komponenata, mrežni modeli udovoljavaju gore spomenutim zahtjevima. Princip je da se cjelokupni uređaj podijeli na dijelove. Dijelovi se zamjenjuju grupama elemenata mreže koji modeliraju njihove funkcije, te se spajaju dijelovima mreže koji opisuju njihovu povezanost. Modeli dijelova ne moraju biti mrežni modeli.

Ovako koncipiran matematički model može poslužiti kao osnova za CAD sistem, koji će omogućiti da projektanti u samoj fazi razvoja provjere veliki broj varijanti za relativno kratko vrijeme.

Transformatori u protueksplozivnoj izvedbi tipičan su primjer složenog sistema (slika 1). Aktivni dio transformatora koji se sastoji od jezgre i namota zatvoren je u neprodorno kućište koje omogućava upotrebu u prostorima gdje se mogu pojaviti eksplozivni plinovi



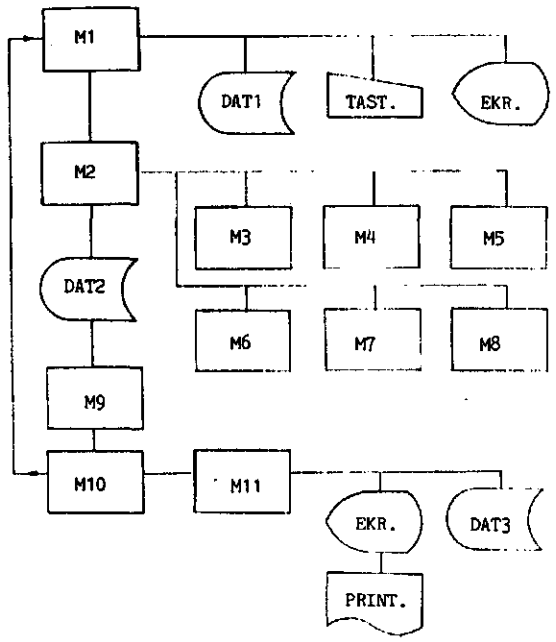
Slika 1. Poprečni presjek transformatora u protueksplozivnoj izvedbi 315 kVA: 1 - kućište, 2 - jezgra, 3 - niskonaponske sekcije, 4 - barijera, 5 - aksijalni rashladni kanali, 6 - visokonaponske sekcije.

ili zapaljiva prašina. U vodičima namota razvijaju se gubici koji su posljedica pretvaranja električne energije u toplinsku.

Namoti su podijeljeni na sekcije, koje su međusobno odvojene aksijalnim rashladnim kanalima čiji raspored i dimenzije treba da spriječe pojavu lokalnih pregrijavanja. Povećanjem snage ovih transformatora, problemi u vezi sa zagrijanjem postaju sve izraženiji, pa predikcija utjecaja pojedinih zahvata na zagrijanje, korištenjem CAD sustava, skraćuje vrijeme potrebno za razvoj novog proizvoda.

2. PRORAČUN ZAGRIJANJA U STACIONARNOM STANJU

Programski sustav za proračun zagrijanja u stacionarnom stanju (slika 2.) omogućuje interaktivni rad, gdje su sve radnje u vezi s formiranjem i rješavanjem matematičkog modela koji odgovara određenoj konfiguraciji u potpunosti automatizirane [1]. Funkcije pojedinih cjelina su:

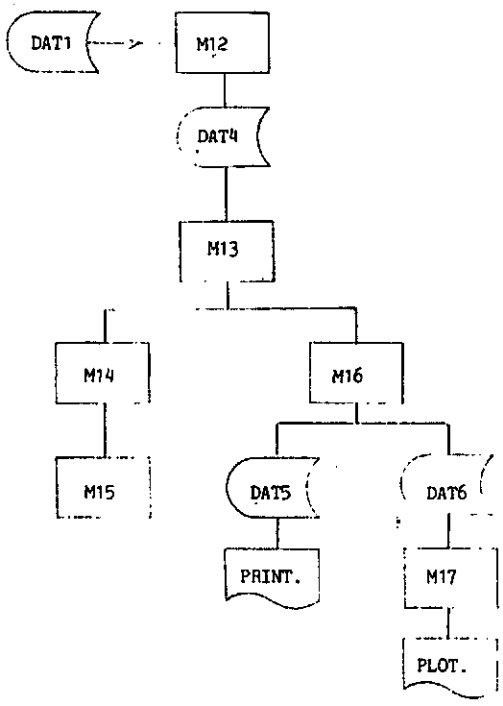


Slika 2.

M1 - Interaktivni program za vođenje kroz proračun. Omogućuje korisniku da odabiranjem ponuđenih varijanti formira konfiguraciju transformatora čije zagrijanje želi proračunati.
DAT1 - Sadrži najneophodnije podatke o transformatoru u obliku pogodnom za korisnika.
M2 - Na osnovu matematičkih modela pojedinih dijelova i njihovih veza (M3 - M8) vrši se popunjavanje datoteka DAT2 i DAT3.
DAT2 - Sadrži sve potrebne informacije o povezanosti mreže koja odgovara odabranoj konfiguraciji transformatora. Sadrži geometrijske veličine i fizikalne konstante materijala dijelova koji sačinjavaju transformator.
M3, M4, M5, M6, M7, M8 - Djelomični modeli: jezgre, sekcije, kućišta, prijenosa topline konvekcijom i zračenjem, veza između djelomičnih modela.
M9 - Formiranje sustava nelinearnih algebarskih jednadžbi koje opisuju zagrijanje transformatora u stacionarnom stanju
M10 - Rješavanje sistema nelinearnih algebarskih jednadžbi
M11 - Obrada i prezentacija rezultata proračuna
DAT3 - Sadrži rezultate proračuna u obliku pogodnom za korisnika.

3. PRORAČUN DINAMIKE ZAGRIJANJA

U svrhu pravilnog dimenzioniranja rashladnog sistema rudničkih transformatora u protueksplozivnoj izvedbi potrebno je poznavati vremenski tok promjene temperatura u raznim pogonskim uvjetima kao što su: preopterećenje, kratki spoj, isprekidani pogon, itd. U ovim slučajevima važno je znati koliko dugo transformator može podnijeti određeno opterećenje, odnosno koliko se najviše smije opteretiti pri određenoj intermitenciji, a da zagrijanje ostane u dopuštenim granicama [2]. Na slici 3. je prikazan dio CAD sustava koji omogućuje proračun dinamike zagrijanja. Funkcije pojedinih cjelina su:

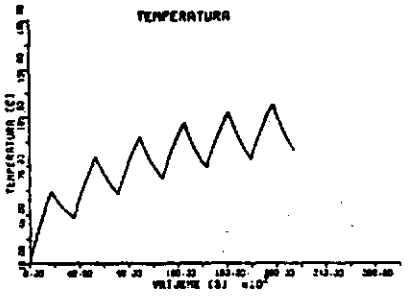


Slika 3.

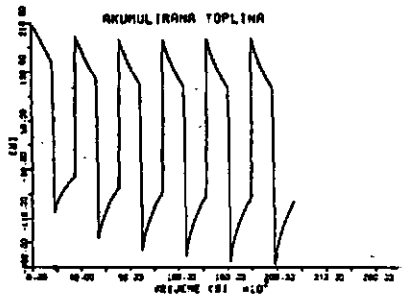
M12 - Formiranje odgovarajuće mreže za proračun dinamike zagrijanja
DAT4 - Sadrži podatke o konfiguraciji mreže, geometriji dijelova, toplinskim kapacitetima dijelova, početnim i rubnim uvjetima
M13 - Formiranje sistema diferencijalnih jednadžbi prvog reda koje opisuju dinamiku zagrijanja
M14 - Rješavanje sistema diferencijalnih jednadžbi prvog reda
M15 - Formiranje desne strane sistema jednadžbi, računanje derivacija
M16 - Tokom integracije vrši se obrada i spremanje rezultata u odabranim točkama u datoteke DAT5 i DAT6
DAT5 - Formatizirana datoteka s osnovnim rezultatima proračuna
DAT6 - Neformatizirana datoteka koja služi za vezu s grafičkim dijelom programa
M17 - Program interaktivnog karaktera za iscrtavanje rezultata na inkrementalnom crtaču.

Na slici 4. prikazan je dio izlaznih podataka proračuna dinamike zagrijanja transformatora 315 kVA za slučaj isprekidanog pogona (a, b, c). Na slikama je prikazana:

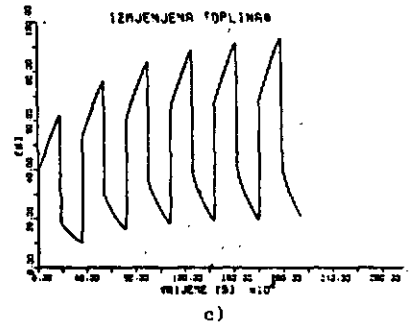
- promjena temperature jedne niskonaponske sekcije
- akumulirana toplina u jedinici vremena
- izmijenjena toplina između nutrine i jedne vanjske plohe sekcije.



a)



b)



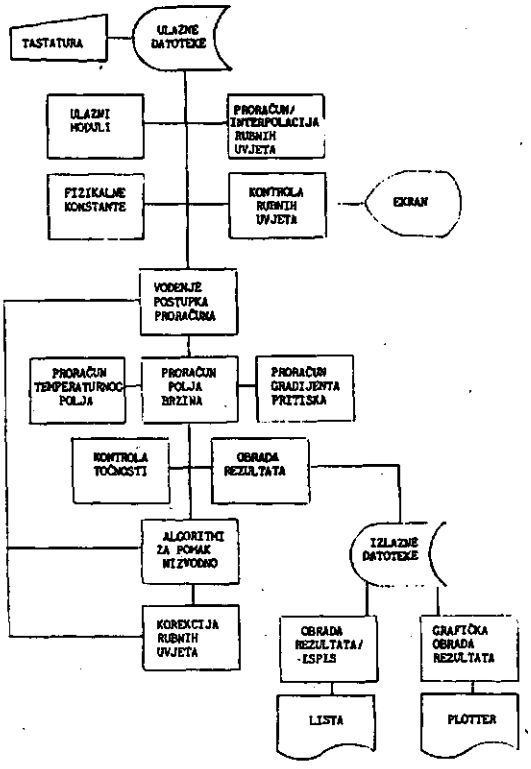
c)

Slika 4.

4. PRORAČUN PRIJELAZA TOPLINE PRIRODNOM KONVEKCIJOM U VERTIKALNIM KANALIMA TRANSFORMATORA

Proračun prijelaza topline prirodnom konvekcijom u aksijalnim rashladnim kanalima (slika 1) transformatora predstavlja ključno pitanje u proračunu zagrijanja. Gibanje zraka koji služi kao rashladni medij prouzrokovano je temperaturnim poljem u samom fluidu pa je zajedno s jednadžbom prijelaza topline potrebno riješiti i jednadžbe strujanja. U proračunu zagrijanja u stacionarnom stanju i proračunu dinamike zagrijanja koji su bazirani na mrežnoj analogiji, konvektivni prijelaz topline se računa preko koeficijenata prijelaza topline odnosno preko odgovarajućih toplinskih vodljivosti. Za detaljnu analizu prijelaza topline u pojedinom aksijalnom kanalu odnosno utjecaja širine, visine i raspodjele temperatura po stijenkama kanala, razvijen je dvodimenzionalni proračun polja temperatura i brzina u vertikalnim kanalima [3].

Na slici 5. prikazana je organizacija sustava i tok postupka proračuna.



Slika 5.

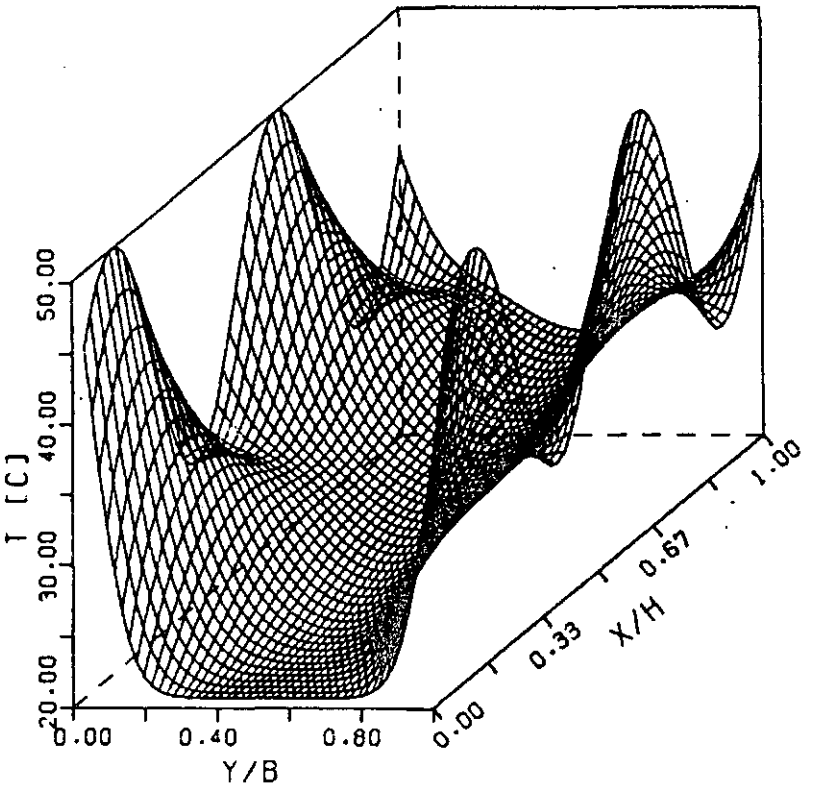
Interaktivni dio sustava omogućuje selektivni dijagramski prikaz pojedinih izračunatih veličina te prikaz cjelokupnih polja temperatura brzina u kosoj projekciji. Za ilustraciju mogućnosti proračuna na slici 6. je prikazano temperaturno polje pri laminarnoj prirodnoj konvekciji u vertikalnom kanalu sa sinusoidalnom temperaturnom raspodjelom na stijenkama kanala.

Osnovni parametri i oznake:

visina kanala H = 0,4 m
širina kanala B = 0,02 m
temperatura stijenki T = 10 sin (12,57X) + 40
temperatura okoline T_{ok} = 20°C
uzdužna i poprečna koordinata X, Y
ukupan broj nepoznanica 60400
pravokutna mreža 400 x 50

5. ZAKLJUČAK

Koncepcija CAD sustava za proračun zagrijanja transformatora u protueksplozivnoj izvedbi omogućuje lako i brzo korištenje mogućnosti koje stoje na raspolaganju. Zbog modularne organizacije sustava kao i pojedinih cjelina proširenje njegovih mogućnosti vrši se jednostavnim dodavanjem novih modula. Projektant ne mora poznavati detalje pojedinih proračuna niti konfiguraciju mreže koja se koristi za modeliranje prijelaza topline. Ukoliko se neki od podataka želi promijeniti procesor za vođenje postupka vraća korisnika na početak i omogućava mu željenu promjenu pa se automatski formira nova mreža koja je u skladu s izmijenjenom konfiguracijom transformatora. Detaljna analiza prijelaza topline u pojedinim kanalima omogućena je proračunom na višem nivou. Interaktivni programi za obradu rezultata omogućuju njihov selektivni grafički prikaz.



Slika 6.

LITERATURA

[1] Zvizdić,D.: "Numerički proračun zagrijanja rudničkih transformatora", Elektrotehnika, ELTHB2, 27(1984)5, 311-320.

[2] Zvizdić,D.: "Prilog analizi prijelaza topline u rudničkim transformatorima s aksijalnim rashladnim kanalima", Strojarstvo, 26(1984)5, 283-288.

[3] Zvizdić,D.: "CAD sustav za proračun prijelaza topline pri strujanjima paraboličkog tipa", Sedmi međunarodni simpozij "Projektiranje i proizvodnja podržani računalom", Zagreb 1985.

NAČRTOVANJE IN PROIZVODNJA IZDELKOV

Tatjana WELZER, Bogomir HORVAT
TEHNIŠKA FAKULTETA MARIBOR

UDK: 681.3.014

POVZETEK - Referat opisuje prednosti CAD in CAM sistemov. To so sistemi, ki omogočajo konstruiranje, oblikovanje, načrtovanje in proizvodnjo s pomočjo računalnika. Opisane so faze razvoja in izdelave izdelka ter tehnična podatkovna zbirka, ki jo potrebujemo za uspešno delo sistemov za načrtovanje in proizvodnjo izdelkov.

DEVELOPMENT AND MANUFACTURING OF PRODUCTS: The paper describes advantages of CAD and CAM systems. This are systems for Computer Aided Design and Computer Aided Manufacturing. Described are development and manufacturing phases of products and tehnical data base, which we need for succes ul work of CAD and CAM systems.

1. UVOD

Izdelke običajno načrtujemo za risalno desko. Ko so odpravljene napake na papirju in je pripravljena vsa potrebna dokumentacija, potrebujemo določen čas za izdelavo modela in njegovo testiranje. Testni rezultati nam omogočajo odkrivanje napak ter popravljanje in spreminjanje parametrov izdelka v obliko primerno za proizvodnjo. Po uspelem preizkusu modela organiziramo redno proizvodnjo izdelka, ki zahteva različna ročna in strojna orodja ter primerno kvalificirano delovno silo.

Zamudno risanje, odpravljanje napak, testiranje, izdelava modela in uvajanje izdelkov v proizvodnjo so problemi, ki najbolj izstopajo in povzročijo, da izdelek od ideje do izvedbe že zastari. Mnogo teh problemov uspešno rešujejo CAD in CAM sistemi oziroma CAD/CAM sistemi.

2. CAD IN CAM

CAD (Computer Aided Design) sistem omogoča konstruiranje, oblikovanje in načrtovanje izdelkov s pomočjo računalniške opreme. Računalniku pri konstruiranju prepuščamo predvsem rutinske naloge. Intenzivnost uporabe računalnika je odvisna od obsega teh nalog. CAD sistem sestavljajo miniračunalnik, masovni pomnilnik, grafični terminal, risalna deska z elektronskim pisalom in sistemska programska oprema. Celoten sistem je načrtovan tako, da omogoča uspešno komuniciranje z uporabnikom. CAM (Computer Aided Manufacturing) je računalniško podprta proizvodnja. Sistem sestavljajo računalniško podprti numerično krmiljeni (NC) stroji in roboti, ki predstavljajo najvišjo stopnjo avtomatizacije.

Osnovna prednost sistemov CAD/CAM je v tem, da odpravljajo pregrado med razvojem izdelka na risalni deski in njegovo proizvodnjo. Omogočajo nam skoraj takojšnjo prilagajanje proizvodnje novih izdelkov, istočasno pa nam nudijo:

a) Kvaliteto - poveča se kvaliteta izdelka, tako v načrtovanju kakor tudi pri izvedbi. Z uporabo CAD lahko pri konstrukciji izdelka raziščemo več možnih modelov. Natančno lahko raziščemo kritične točke in dosežemo večjo zanesljivost.

b) Za izvedbo projekta porabimo mnogo manj časa, kot pri klasičnem načinu. Načrti so boljši, manj je nepotrebnih podatkov, testi modelov ob spremembi parametrov so hitrejši.

c) Izboljša se medsebojna povezanost različnih programov. Izhod nekega programa je lahko istočasno tudi vhod nekega drugega programa.

d) CAD/CAM sistem nam v nekaterih primerih edini omogoča reševanje zapletenih in obsežnih problemov kot celote.

e) Pri posameznih projektih v klasičnem primeru sodeluje mnogo različnih strokovnjakov za posamezna področja. Z uporabo CAD/CAM sistema pa se število sodelavcev lahko močno zmanjša.

3. RAČUNALNIŠKO PODPRTE FAZE RAZVOJA IN IZDELAVE IZDELKA

Računalniško podprt razvoj in izdelavo izdelka izvedemo v več korakih:

a) V fazi priprave projektne dokumentacije oblikujemo osnutek projekta izdelka. V njem podamo tehnične podatke in zahteve o izdelku, njegove možne rešitve, prednosti in slabosti. Projektna dokumentacija vsebuje tudi finančno oceno izdelka stroške in čas izdelave, potrebna ročna in strojna orodja ter primerno kvalificirano delovno silo za razvoj in izdelavo izdelka. Oblikovanje takšnega osnutka nam omogoča ustrezen informacijski sistem. Sistem zbira kvalitetne in najnovejše podatke (o materialih, standardih, dimenzijah, lastnostih, značilnostih, meritvah in kvaliteti) skrbi za njihov prenos, obdelavo, shranjevanje in dostavljanje uporabnikom.

b) Konstruiranje je proces obdelave podatkov, katerih končni rezultat je izdelek oziroma ustrezna informacija o njem. Bistvo faze je iterativnost, saj jo lahko večkrat ponovimo in tako dobimo za isti izdelek več možnih rešitev. Fazo konstruiranja sestavljata še dve podfazi: priprava koncepta in snovanja, nalogi, ki

ju v večini primerov rešuje konstrukter. Da je njegovo delo uspešno, mora razpolagati z množico informacij. Informacije za posamezne izdelke in materiale so lahko ekonomske ali tehnične. Dobimo jih iz ustrezne podatkovne zbirke in iz drugih informacijskih sistemov. Pri tem zahtevamo aktualnost (podatki morajo biti kar se da novi) in natančnost.

c) Rezultat sinteze oziroma konstrukcije je želeni model izdelka. Preizkušamo lahko abstraktni (teoretični model) ali fizični model (prototipni model, pilotski model, pomanjšani model). Pred izdelavo modela poznamo tehnične zahteve in karakteristike večine agregatov in elementov (agregati so sestavne enote izdelka, elementi pa so sestavne enote le-teh). Ob njihovi združitvi pride do interakcije (medsebojnega vpliva združenih agregatov in elementov). To lahko povzroči spremembo začetnih karakteristik in karakteristiko celotnega prototipa. Zato dejanske karakteristike modela izmerimo in analiziramo s pomočjo različnih testnih programov. Vhodne podatke spreminjamo tako kot se spreminjajo v realnih pogojih delovanja in opazujemo njihove izhodne oziroma končne rezultate, ki jih dobimo v obliki diagramov, izračunov in opisov. Z ustrezno računalniško materialno in programsko opremo lahko rezultate izrišemo na zaslon grafičnega terminala ali na papir. Pričakovani rezultati nas vodijo v izdelavo izdelka in v pripravo dokumentacije. V nasprotnem primeru pa fazo izdelave in preizkus modela izdelka ponovimo.

d) Izdelava izdelka predstavlja konec razvoja izdelka in začetek njegove serijske proizvodnje, ki jo lahko izvajamo na različne načine. Klasična industrijska proizvodnja poteka za preprostimi "neinteligentnimi" stroji in z velikim deležem žive delovne sile. Pri težjih, nevarnih in monotonih opravilih pa vedno pogosteje uporabljamo numerično krmiljenje - NC stroje. Ob nadaljnji avtomatizaciji proizvodnje lahko pričakujemo, da bodo živo delovno silo zamenjali roboti. Le-ti se v industriji pojavljajo na delovnih mestih s težkimi delovnimi pogoji, kot manipulatorji za prenašanje posameznih agregatov in elementov, za varjenje, barvanje in kontrolo.

e) Priprava dokumentacije daje izhodne informacije o opravljenih nalogah v predhodnih fazah. Dokumentacijo o izdelku naredimo na osnovi predhodno postavljenih tehničnih zahtev za izdelek in iz rezultatov, ki smo jih dobili pri analizi in testiranju modela. Vsebuje pa tudi tehnične in tehnološke podatke o izdelku in njegovih elementih.

4. ZBIRKA TEHNIČNIH PODATKOV

Za izvedbo opisanih nalog potrebujemo kopico podatkov, karakteristik in standardov o posameznih agregatih in elementih. Podatke hranimo v "informacijskem skladišču", ki ga sestavlja ena ali več podatkovnih zbirk (uporabljamo tudi ime tehnična podatkovna zbirka).

Podatkovno zbirko definiramo kot neredundantno zbirko vzajemno povezanih podatkov, kot množico višjih logičnih zbirk ali kot zbirko podatkov, ki so medsebojno povezani brez nepotrebne redundance in omogočajo optimalno izvajanje najrazličnejših aplikacij. Po svoji strukturi so podatkovne zbirke lahko hirearhične, mrežne ali relacijske.

Hirearhična zbirka je sestavljena iz segmentov. Najvišji nivo ima le en segment - temeljni segment ali izvor. Razen izvora, imajo vsi ostali elementi nad sabo en sam temeljni segment, ki pa je lahko element na višjem nivoju ležečega temeljnega segmenta. Drevesna struktura je lahko homogena, heterogena, uravnotežene ali neuravnotežena.

Za razliko od drevesne strukture (kjer obstaja za posamezen nivo le en temeljni segment) imamo lahko v mrežni podatkovni zbirki več temeljih segmentov, kar nam omogoča poljubne povezave med posameznimi elementi. Za mrežo lahko tudi rečemo, da je homogena ali heterogena, odvisno od tipov segmentov, ki jo sestavljajo.

Zanimivo povezavo med elementi oziroma odvisnost med posameznimi segmenti nam daje relacijska zbirka podatkov. Organizirana je v obliki tabele, kar je ugodno predvsem za uporabnika, saj lahko odvisnost med segmenti spremljamo v obeh smereh direktno, medtem ko moramo pri mreži ali drevesu posamezne povezave ugotavljati od nivoja do nivoja.

Pri projektiranju in izdelavi zahtevnejših izdelkov pa potrebujemo več računalnikov. Vsak računalnik skrbi za načrtovanje in izdelavo določenega elementa ali skupine elementov. Za uspešno končno izvedbo izdelka, moramo računalnike povezati med seboj. Na računalniški mreži zgradimo avtomatiziran lokalni tehnično informacijski sistem, ki uporabnikom omogoča kvalitetno delo in ustrezne medsebojne komunikacije.

4.1. KLASIFIKACIJA PODATKOV

Zahtevna naloga pri oblikovanju tehnične podatkovne zbirke je zbiranje podatkov. Raztreseni so po najrazličnejših mestih (v literaturi, priročnikih, dokumentaciji, zahtevah uporabnikov, protokolih, ...). Zato je izredno težko vse te informacije neredudantno zbrati, saj so nekateri izmed naštetih virov večkrat nenatančni in neurejeni.

Zbiranju podatkov sledi klasifikacija, pri kateri podatke sestavljamo v homogene kategorije ali skupine, na osnovi določenih skupnih značilnosti (dimenzija, material, oblika, barva, fizikalne in kemijske lastnosti). Pravilno klasifikacijo dosežemo z uporabo naslednjih pravil:

- natančno definiramo posamezne karakteristike agregatov in elementov,
- določimo logično strukturo podatkov,
- zagotovimo nadaljnje širjenje homogenih kategorij in
- upoštevamo pravila za klasifikacijo.

Po uspeli razvrstitvi oziroma združitvi podatkov v homogene skupine je naslednji korak identifikacija. Vsaki skupini podatkov in vsakemu podatku moramo prirediti ustrezno šifro ali ključ, ki nam omogoča, da podatek vedno hitro in enostavno najdemo v tehnični podatkovni zbirki. Pri izbiri ključa imamo veliko možnosti, vendar moramo upoštevati omejitve, ki izhajajo iz morebitnih zahtev uporabniških programov in sistema za upravljanje s podatkovno zbirko. Zato ključ izberemo tako, da bomo uporabili čim manj prostora.

5. ZAKLJUČEK

Pri računalniško podprti proizvodnji prihranimo mnogo časa pri konstrukciji in izdelavi izdelka. Preizkusimo oziroma testiramo lahko tudi več možnih rešitev in izberemo najboljšo med njimi, kvalitetna zbirka tehničnih podatkov predstavlja zelo dobro osnovo za dokumentacijo in nadaljnje načrtovanje. Računalniško podprti sistemi za načrtovanje in proizvodnjo izdelkov bodo popolnoma zaživeli ko bomo odpravili nekatere kritične točke. Izboljšati

moramo obstoječe informacijske sisteme, zbiranje in obnavljanje podatkov, uvesti je potrebno sodobno opremo (grafični terminali, risalne deske z elektronskim pisalom, sodobni risalniki), izpopolniti je potrebno robote, dobrodošla pa bo tudi pomoč umetne inteligence.

Pri uvajanju CAD/CAM sistemov, moramo upoštevati tudi ekonomski in socialni vpliv teh sistemov v industriji. Hitrejša in avomatizirana proizvodnja zmanjšuje namreč število potrebnih delavcev, istočasno pa zahteva tudi velika sredstva za popolno uveljavitev sistema in posodobitev proizvodnje.

6. LITERATURA

1. Carl Machover, Robert E. Blauth: The CAD/CAM Handbook Computervision, Bedford 1980

2. Martin Prašnički: Model sistema za avtomatizirano konstruiranje sklopov vozil z računalniško grafiko in miniračunalnikom, Magistrsko delo, VTŠ Maribor 1981

3. Interface challenges loom for CAM in computer-controlled automation, Electronic Design, November 10 1983, str. 109-127

RAČUNALNIŠKE MREŽE

COMPUTER NETWORKS

KOMUNIKACIJSKI KRMILNIK

Rado SLATINEK, Bogomir HORVAT, Nenad ČRNKO
TEHNIŠKA FAKULTETA MARIBOR

UDK: 681.3.007

POVZETEK - Članek opisuje zgradbo in delovanje mikroprocesorsko podprtega komunikacijskega krmilnika računalniške delovne postaje TK 68000. Materialno opremo modula sestavljajo mikroprocesor 68000, štirje periferni vmesniki 8530 SCC, časovnik 6840, 128/512 kB DRAM, 4/16/32/64 kB EPROM, vezje 8038 FIO, dodeljevalnik lokalnega vodila in vmesnik za povezavo z multipleksiranim sistemskim vodilom. Krmilnik omogoča vzpostavljanje povezav preko osmih asinhronih ali sinhronih serijskih komunikacijskih kanalov z raznovrstno terminalsko opremo ter povezavo računalniškega sistema v lokalne in javne komunikacijske mreže za prenos podatkov.

COMMUNICATION CONTROLLER: The paper describes a functional structure and operating of a microprocessor based communication controller, which is part of the computer workstation TK 68000. The controller is composed of the 68000 microprocessor, four 8530 serial communication controllers, the 6840 timer, 128/512 kB of DRAM, 4/16/32/64 kB of EPROM, 8038 input/output interface unit,an arbiter and an interface to the host's multiplexed sytem bus. The controller enables connection with different terminal equipment and with local and public data networks on eight asynchronous and synchronous communication chanels.

1. UVOD

V današnjem času, ko prehajamo na distribuirano obdelavo podatkov, je potreba po komuniciranju med računalniškimi sistemi čedalje večja. Povezovanje računalniške opreme (veliki sistemi, procesni računalniki, delovne postaje, osebni računalniki itd.) različnih proizvajalcev je zaradi neenotnih komunikacijskih protokolov posameznih sistemov zelo oteženo. Univerzalni komunikacijski modul delovne postaje rešuje komunikacijske potrebe uporabnika na nivoju terminal - delovna postaja, hkrati pa omogoča izmenjavo podatkov s preostalimi računalniškimi sistemi v organizaciji in izven nje.

Komunikacijski krmilnik (KK) je zaključen mikroračunalniški sistem, ki lahko prevzame od centralnega računalnika večino komunikacijskih opravil. Omogoča pretok podatkov in procesiranje mrežnih protokolov na nižjih nivojih med računalnikom in nanj priključenimi napravami. KK je načrtovan kot sestavni del delovne postaje TK 68000 z večuporabniškim operacijskim sistemom, vendar lahko deluje tudi kot samostojna enota z lastnim operacijskim sistemom.

Pri načrtovanju univerzalnega modula komunikacijskega procesorja smo upoštevali cenovno optimizacijo ter dosegljivost materialne opreme, nizke stroške razvoja in predvideli široke možnosti uporabe komunikacijskega krmilnika. Predvideno delovno okolje zahteva več serijskih komunikacijskih kanalov, od katerih sta vsaj dva sinhrona (SNA/SDLC, X.25 ali medsebojna povezava računalnikov TK 68000) preostali pa asinhroni za priključitev terminalov in druge periferne opreme. Krmilnik je mikroračunalniški modul z osmimi serijskimi komunikacijskimi kanali za priključitev različne računalniške, terminalske in periferne opreme na delovno postajo. Zasnovan je tako, da z ustrezno programsko podporo omogoča povezavo delovne postaje v lokalno računalniško in javno podatkovno mrežo ter pod določenimi pogoji tudi na PABX naročniško telefonsko centralo.

2. ZGRADBA KOMUNIKACIJSKEGA KRMILNIKA

Krmilnik je povezan s centralnim procesorjem preko sistemskega vodila. Časovno multipleksirano naslovno-podatkovno sistemsko vodilo (24-bitni naslovni in 16-bitni podatkovni del) povezuje vse module računalniškega sistema TK 68000. V osnovni konfiguraciji vsebuje sistem procesorski modul, DRAM modul, komunikacijski krmilnik ter modula za priključitev zunanjega masovnega pomnilnika in konzole. V razširjeni sestavi ima računalnik še DMA modul, dodatni DRAM pomnilnik in module za različno periferijo. Slika 1 prikazuje povezavo modulov preko sistemskega vodila.

Materialno opremo KK grobo delimo v dva velika funkcionalna sklopa:

- vmesnik za vodilo TK 68000 in
- komunikacijski procesor.

Vmesnik je zasnovan tako, da opravlja naslednje naloge:

- skrbi za pravilno odzivanje, ko glavni procesor (GP) ali DMA naslovni KK,
- demultipleksira naslovno-podatkovno vodilo TK 68000,
- omogoča pravilno dodeljevanje lokalnega vodila,
- pri bralnem ciklu generira pariteto podatkov, ob vpisnem pa jo preverja,
- signalizira napako pri prenosu podatkov med GP in KK ter dvojno napako na lokalnem vodilu,
- krmilni naslovni števnik.

Sestavljajo ga:

- demultipleksor,
- naslovni dekodirnik,
- naslovni števnik,
- vezje 8038 FIO, ki vsebuje 128 bytov globoko FIFO vrsto in dva sporočilna registra.

Komunikacijski procesor je zgrajen iz naslednjih sklopov:

- lokalnega procesorja 68000,
- naslovnega dekodarnika in povezovalnih sestavov,
- pomnilnika EPROM kapacitete 4/16/32/64 kB,
- dinamičnega RAM-a 128/512 kB,
- registra prioritete,
- perifernih vmesnikov 8530 SCC,
- merilnika časovnih intervalov PTM 6840 in
- linijskih vmesnikov po standardih RS-232 ali RS-422.

Slika 2 prikazuje zgradbo komunikacijskega krmilnika.

3. DELOVANJE KOMUNIKACIJSKEGA KRMILNIKA

Sistemsko vodilo je povezano z lokalnim preko naslovnega števnika in vmesnikov. Dodeljevalnik (arbiter) skrbi za pravilen prenos podatkov med vodiloma, tako da lokalno vodilo zaseda hkrati le en procesor. Na vodilu ne sme priti do trkov prenosov različnih procesorjev, zato dodeljevalnik izmenično dodeljuje vodilo GP in KP, kadar oba zahtevata dostop na lokalno vodilo. S pomočjo sporočilnih registrov elementa FIO se GP in KP obveščata o pomembnih dogodkih in potrebnih ukrepih. FIFO vrsta je usmerjena od lokalnega k TK vodilu. Namenjena je za prenos znakov s periferije h glavnemu procesorju. To je ugodna rešitev, ker omogoča znakovnemu prenosu prioriteto nad blokovnim, kar zagotovi dobre odzivne čase pri interaktivnem delu. FIFO obenem zagotovi časovno enakomeren priliv znakov, zaradi tega GP-ju ni več potrebno togo slediti dogodkom na znakovnih komunikacijskih kanalih. KK ima samo en FIO element zaradi prostorske in cenovne optimizacije vezja, ne da bi se zaradi tega bistveno zmanjšale njegove zmogljivosti.

Za blokovni prenos ima GP neposreden pristop na lokalno vodilo. Če bi imel možnost naključnega naslavljanja, bi moral biti ves naslovni prostor KP viden GP-ju kot del njegovega naslovnega prostora. Ker je mogoče v sistem vključiti več KK, bi vsi skupaj zasedli dobršen del naslovov, ki so sicer namenjeni pomnilnikom. Mehanizem okna zmanjša potrebno število naslovov na najmanjšo možno mero.

KK zavzema v naslovnem polju le 16 naslovov:

H'FFXXX0 - podatkovni register FIO
H'FFXXX2 - kontrolni register FIO
H'FFXXX4 - MSW - zgornji del naslovnega števnika
H'FFXXX6 - LSW - spodnji del naslovnega števnika
H'FFXXX8 - R/W - okno k lokalnemu vodilu
H'FFXXXA - RI/WI - okno k lokalnemu vodilu
H'FFXXXC - RI/WI - okno k lokalnemu vodilu
H'FFXXXE - RD/WD - okno k lokalnemu vodilu

Slika 3 prikazuje naslovna področja računalniškega sistema TK 68000 in komunikacijskega krmilnika.

FIO zavzema prve štiri naslove (v resnici se nahaja podatkovni register na naslovu H'FFXXX1, kontrolni pa na H'FFXXX3). Na naslovnih H'FFXXX4 do H'FFXXX7 je naslovni števnik (MSW, LSW). Vanj je potrebno vpisati vrednost, ki ustreza želenemu naslovu na lokalnem vodilu. To lahko storimo npr. z inštrukcijo

MOVE.L ADR,$FFXXX4.

Z izbiro enega ali dveh naslovov (MOVE.B ali MOVE.W) izmed preostalih osem, generiramo na lokalnem vodilu naslov, ki je določen z vsebino naslovnega števnika. V odvisnosti od izbire adresnega načina (MOVE.B, MOVE.W ali MOVE.L) in naslova v oknu ostane vsebina naslovnega števnika nespremenjena, se poveča ali zmanjša za 2 ali 4. Okno na naslovu H'FFXXX8 kaže vedno na eno lokacijo, dokler GP ne spremeni vsebino naslovnega števnika. Okna na naslovih od H'FFXXXA do H'FFXXXE omogočajo sekvenčni pristop, kjer se po vsakem prenosu podatka skozi okno vrednost naslovnega števnika ustrezno spremeni. Ob naslovitvi enega od oken se sproži proces dodeljevanja lokalnega vodila. Pri tem velja princip "sprosti na zahtevo". Če bi uporabili princip "sprosti, ko si opravil", bi bilo ogroženo delovanje operacijskega sistema GP.

Vsakič, ko GP naslovi lokalno vodilo, mora v povprečju čakati nekaj več kot 1/2 časa izvajanja cikla na vodilu. Dodeljevanje lokalnega vodila smo optimirali tako, da se lokalni procesor izklopi potem, ko je postoril vsa opravila. GP tedaj ob naslovitvi KK takoj doseže želeno lokacijo.

Stanje bita v registru prioritete opravil je kriterij za izklop lokalnega procesorja. Potem, ko je končal s tekočimi opravili, vpiše v register prioritete vrednost "0". S tem se izklopi in GP lahko doseže lokalno vodilo brez čakanja. Katerakoli prekinitev povzroči, da se vsebina registra prioritete postavi na "1". To povzroči vklop KP in dodeljevanje vodila "na zahtevo".

Za pravilno delovanje sistema je potrebno uporabiti naslednji programski prijem:

Prekinitvene rutine za vsak dogodek postavijo ustrezno bit, ki pove monitorju, kaj mu je storiti. Poleg teh bitov je potrebno definirati še dodatnega, imenujmo ga INT bit. Le-tega mora postaviti na "1" vsaka prekinitvena rutina. Monitor ciklično preglejuje bite posameznih dogodkov in glede na njih stanje ustrezno ukrepa.

Ob začetku cikla postavi INT bit na "0" in nato nadaljuje s pregledovanjem ostalih bitov. Ko pregleda vse (in seveda konča z opravili), testira še INT bit. Če ima le-ta v tem trenutku vrednost "0", lahko vpiše "0" v register prioritete in s tem izklopi procesor. V nasprotnem primeru mora nadaljevati delo na začetku cikla, ne da bi vpisal prioriteto "0".

4. PROGRAMSKA OPREMA KOMUNIKACIJSKEGA KRMILNIKA

KK ima rezidenčno programsko opremo vpisano v EPROM in programe, ki jih naloži GP. Po vklopu računalnika LP z rezidenčnim testnim programom preveri lokalna vezja (DRAM, SCC, FIO in PTM). Če materialna oprema deluje pravilno, GP naloži komunikacijsko programsko opremo v lokalni DRAM KK. Nato se inicializirajo parametri, ki omogočajo pravilno delovanje LP (sistemske spremenljivke LP) in komunikacijskih kanalov (inicializacija logičnih in fizičnih kanalov). Po inicializaciji vezij (FIO, SCC in PTM) je KK pripravljen za sprejem in oddajo podatkov po komunikacijskih kanalih.

Ker komunikacijski programi niso rezidenčni, lahko hitro in preprosto reinicializiramo posamezni kanal za drugačni način prenosa podatkov. Programe lahko predhodno vpišemo tudi v lokalni EPROM pomnilnik in tako sprostimo del bralno-vpisnega pomnilnika za pretok podatkov. Tedaj ne moremo programsko spreminjati delovanja komunikacijskih kanalov, vendar lahko KK deluje popolnoma samostojno.

Komunikacijsko programsko opremo sestavljajo rutine in povezovalni program (jedro). Procesiranje posamezne rutine se sproži pod določenimi pogoji. Prekinitev sproži eno izmed prekinitvenih rutin. Rutine prenesejo podatke med vmesniki (FIO ali SCC) in vmesnim pomnilnikom (buffer) ali pa sprožijo 'time-out' proceduro (PTM) pri sinhronem prenosu podatkov. Rutine za obdelavo podatkov

omogočajo različne operacije nad podatki in procesiranje kontrolnih znakov in informacije v blokih podatkov (kodiranje/dekodiranje, zaščita podatkov, kontrola pretoka, odmev, ...). Prekladalne rutine premeščajo podatke iz vmesnih pomnilnikov v vrste, v katerih podatki čakajo na prenos v sistem ali v komunikacijski kanal in omogočajo prenos kontrolnih sporočil med LP in GP v obeh smereh. Kontrolna sporočila se izmenjujejo skozi ukazni kanal (sporočilna registra v FIO elementu), preko katerega si LP in GP posredujeta informacije o zasedenosti kanalov oziroma vrst in načinih delovanja. Jedro programske opreme ugotavlja pogoje za skok v posamezno rutino. KK lahko ima dodatne programe za statistično obdelavo pretoka podatkov.

5. PRETOK PODATKOV SKOZI KOMUNIKACIJSKI KRMILNIK

Pretok podatkov skozi KK do procesa v centralnem računalniku poteka skozi znakovni (asinhroni prenos) ali blokovni (asinhroni in sinhroni prenos) kanal. V znakovnem kanalu se podatki prenašajo skozi KK znak za znakom, medtem ko blokovni kanal prenaša bloke podatkov. Pri asinhronem blokovnem prenosu se prenašajo vrstice in pri sinhronem prenosu paketi (podatkovni del HDLC/ /SDLC nabora) podatkov. Prednost blokovnega prenosa je veliko večja propustnost sistema, saj procesor ne izgublja časa z vsakim posameznim podatkom posebej. Bloke podatkov lahko preko sistemskega vodila izredno hitro prenaša tudi DMA krmilnik.

GP ima dve možnosti komuniciranja z napravami na lokalnem vodilu: prenos skozi FIFO vrsto (enosmerni znakovni kanal k GP) in prenos skozi okno (dvosmerni blokovni kanal in enosmerni znakovni kanal k LP).

Prikazan je asinhron znakovni pretok podatkov skozi KK, ki je uporabljen z osnovnim monitorjem za testiranje delovanja KK. Sprejem podatka po znakovnem kanalu sproži prekinitveno rutino (Rv), ki podatek prepiše v vmesni pomnilnik (buffer). Vsak znakovni kanal ima svoj vmesni pomnilnik. Podatki se iz vmesnega pomnilnika prepišejo v FIFO register FIO elementa. FIFO register se uporablja kot vmesni kanal za prenos podatkov k GP. Enakomernejši pretok skozi KK za vse znakovne kanale dosežemo z zaporednim praznenjem vseh zasedenih vmesnih pomnilnikov. Rutina Rf prenese iz vmesnega pomnilnika v FIO le en znak in nato preskoči na naslednji buffer neglede na to, koliko je ta buffer zaseden. Hitri znakovni kanali tako ne morejo "odriniti" počasnejših. Vsak podatek spremlja tudi kontrolna informacija, ki določa, skozi kateri kanal je bil podatek sprejet. V GP rutina Rs prepiše podatek iz FIO v ustrezni sprejemni vmesni pomnilnik, ki je dodeljen posameznemu kanalu. Tu je podatek na voljo operacijskemu sistemu računalnika oz. uporabniškemu programu.

KK ima samo en FIO element, zato mora GP uporabljati pri oddajanju podatkov (rutina Ro) pomnilnik KK kot vmesni kanal med GP in LP. Takšna konfiguracija je izbrana zaradi večjega izhodnega toka podatkov, ki ni časovno kritičen. Kadar želi GP odposlati podatke, skozi okno napolni izhodno vrsto z enim ali več znaki. LP dobi preko ukaznega kanala (sporočilni register FIO) sporočilo o zasedenosti vrste. Rutina Rz sproži rutino Ri, ki prenese podatke v ustrezni periferni vmesnik SCC oz. komunikacijski kanal. Ko je vrsta prazna, LP preko ukaznega kanala sproži GP, da lahko ponovno napolni vrsto. Rutina Rp omogoči, da Ro lahko izprazni oddajni vmesni pomnilnik.

Slika 4 prikazuje preprost pretok podatkov skozi znakovne asinhrone vhodno/izhodne kanale.

Paketi podatkov (blokovni kanal) se po sprejemu shranijo v vmesnem pomnilniku. Vsak fizični kanal ima svoj buffer za en paket. Poln sprejemni buffer se mora čimprej izprazniti, da je pripravljen na sprejem novega paketa. Paketi se prenesejo v vrsto, kjer čakajo, dokler se vmesni kanal ne sprosti. Med prenosom LP obdela celotni blok in ustrezno ukrepa. Vmesni kanal je del pomnilnika, iz katerega GP prenese blok v svoje delovno področje oziroma višjim nivojem operacijskega sistema. Preko ukazanega kanali si procesorja izmenjujeta sporočila o zasedenosti vrste. Odpošiljanje blokov podatkov poteka po enakem principu kot sprejem. Splošen pretok podatkov skozi KK prikazuje slika 5.

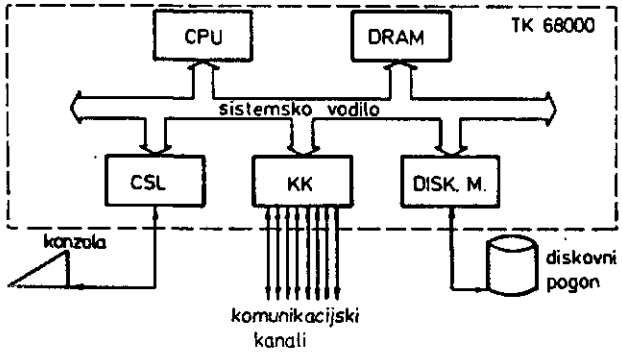
6. ZAKLJUČEK

V zasnovi KK smo upoštevali različne pogoje obratovanja krmilnika, zato pričakujemo, da bo KK uspešen komunikacijski sestav z izredno široko uporabo, ne samo v sklopu računalnika TK 68000, temveč tudi kot samostojna enota.

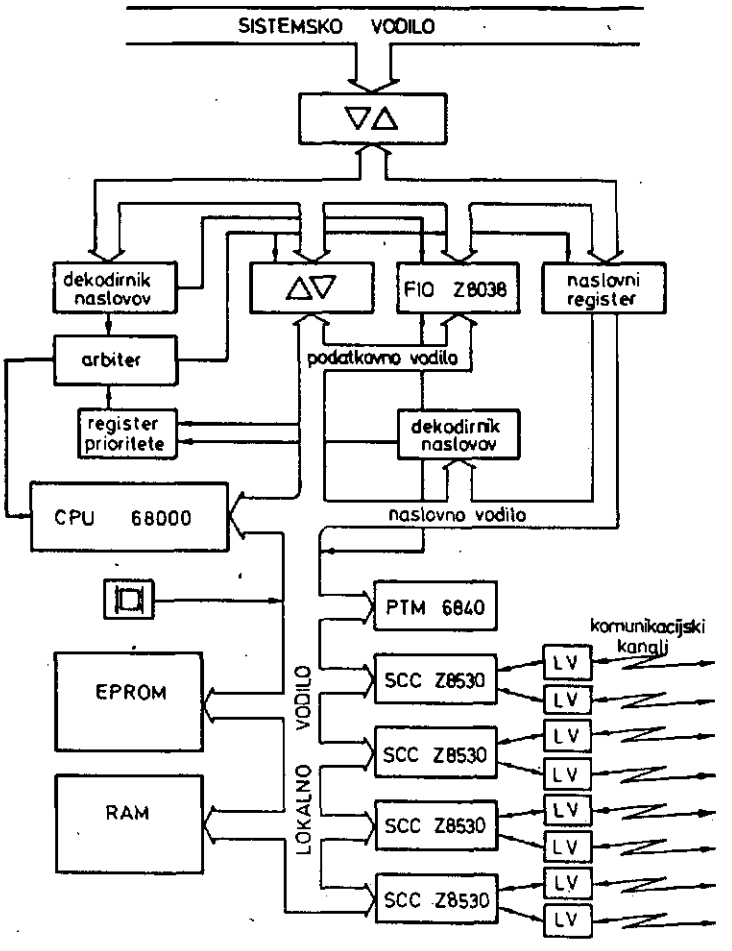
7. REFERENCE

(1) I. Dolinšek, P. Knaflič: "Specifikacija zahtev do inteligentnega komunikacijskega krmilnika", interni dokument ISKRE TELEMATIKE, dec. 1984, str. 12, Ljubljana,

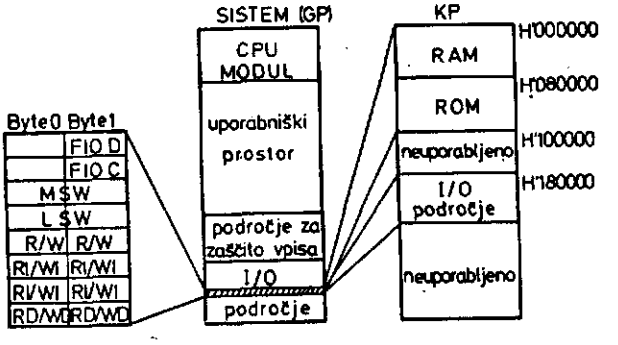
(2) R. Slatinek, N. Črnko: "Poročilo o 1. fazi razvoja komunikacijskega krmilnika", delovno poročilo za ISKRO TELEMATIKO, jan. 1985, str. 42, Maribor.



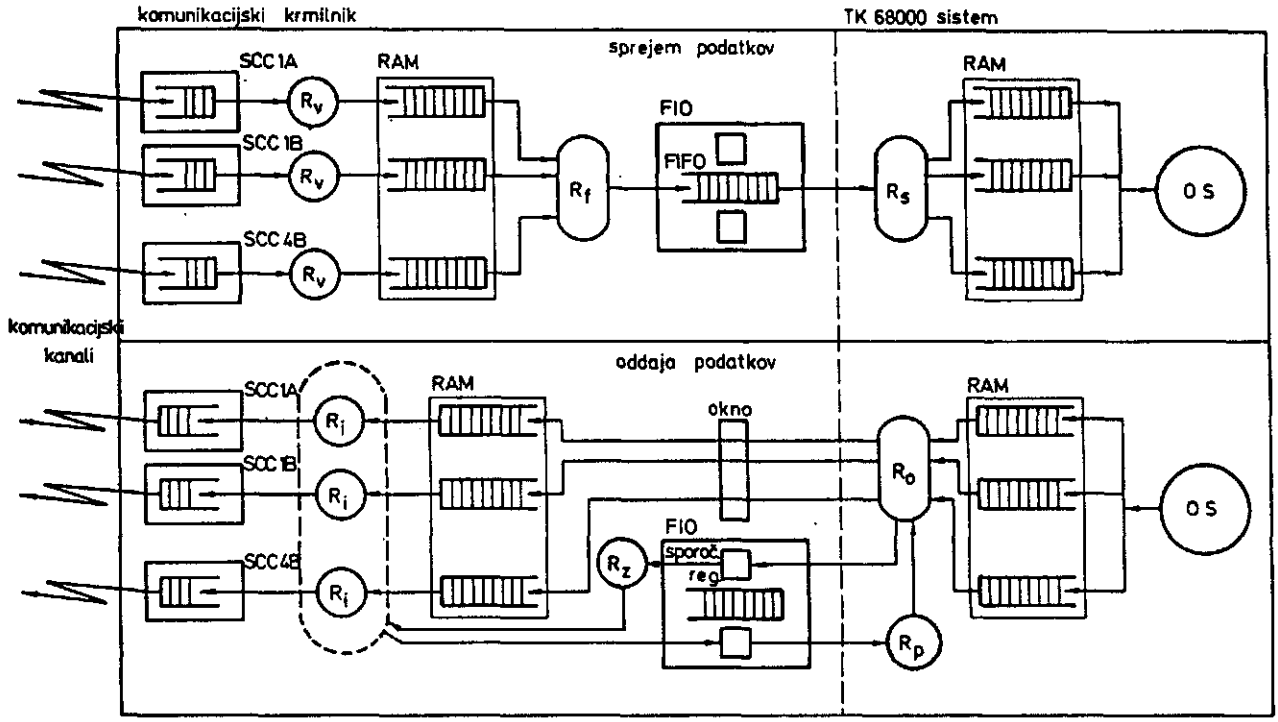
Slika 1: Osnovni moduli računalnika TK 68000



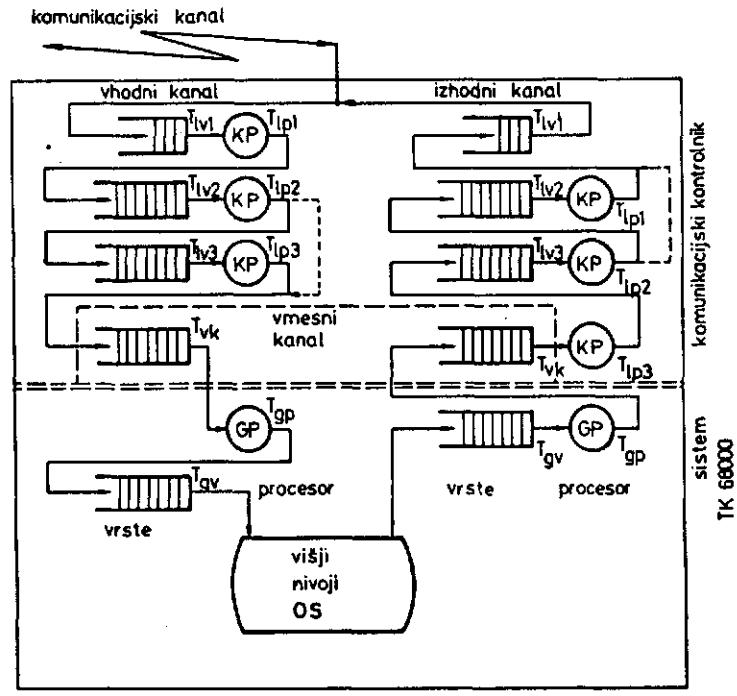
Slika 2: Zgradba komunikacijskega krmilnika



Slika 3: Razporeditev naslovov sistema in KK



Slika 4: Znakovni asinhroni vhodno/izhodni kanali



Slika 5: Pretok podatkov skozi KK

NEKATERE IZKUŠNJE PRI DELU Z RAČUNALNIŠKIMI MREŽAMI

Davor Šoštarič,
UNIVERZA V MARIBORU
RAČUNALNIŠKI CENTER
Maribor, Jugoslavija

UDK: 681.3.007

NEKATERE IZKUŠNJE PRI DELU Z RAČUNALNIŠKIMI MREŽAMI

V referatu so prikazane nekatere naše izkušnje pri izgradnji in delovanju računalniške mreže Univerze v Mariboru. Prikazane so nekatere dobre lastnosti in slabosti, ki smo jih spoznali v času delovanja, poleg tega pa tudi nekatere težave, ki smo jih morali premagati v izgradnji.

EXPERIENCES USING A COMPUTER NETWORK

In this article, we describe some of our experiences while installing, using, and maintaining our computer network at the University of Maribor. The article tells about the problems we had to overcome when installing the network and gives our opinion of the good and bad features of it.

Uvod

Potrebe članic Univerze v Mariboru so v preteklosti narekovale uporabo in nabavo različnih računalniških sistemov. Tako smo na posameznih šolah do leta 1983 srečali računalnik DELTA 400 ter po dva računalnika DELTA 340 in ISKRADATA C-18, poleg tega pa so še uporabljali DEC-10 na Univerzi v Ljubljani in CYBER 72/172 na Republiškem računskem centru. Vsaka šola je sama zase iskala svoje poti v računalništvo in s tem seveda tudi po svoje skrbela za svojo računalniško opremo. Tu je prihajalo do večjih razkorakov, tako v programski kot tudi v specialni strojni opremi.

Ko je bil ustanovljen skupni univerzitetni računalniški center, je bila ena od njegovih temeljnih nalog tudi posodobitev in povezava vseh teh računalnikov. Pri tako različni paleti želja, potreb in parcialnih zmogljivosti je bilo seveda težko spraviti vse na skupni imenovalec. Brez nekaterih temeljnih, morda tudi bolečih rezov, ni šlo. Na podlagi različnih presoj smo se odločili za izgradnjo takega računalniškega sistema, ki bi posameznim članicam nudil dovolj avtonomne moči, obenem pa omogočil povezavo sistemov v integrirano računalniško mrežo. Našim namenom in možnostim je najbolje ustrezal koncept Digitala oziroma ISKRA DELTE, tako da smo kot centralni računalniški sistem instalirali računalnik DELTA 4850, sistem ISKRADATA C-18 pa zamenjali z DELTO 340 in DELTO 644. Po zagotovitvah proizvajalcev se dajo vsi ti sistemi enostavno povezati v mreže, saj vsi podpirajo DECNET - Digitalov software za računalniško mrežo, poleg tega pa se je možno z različnimi emulatorji priključiti tudi k večjim računalnikom drugih tipov (IBM, CDC,..).

Današnja topologija računalniške mreže Univerze v Mariboru

Na univerzitetnem računalniškem centru imamo tri računalnike DELTA 4850. Dva od teh treh sta bila šele nedavno instalirana in še nista povezana v mrežo. Na posameznih šolah je stanje tako:

- Tehniška fakulteta - VTO Strojništvo: DELTA 340
- Tehniška fakulteta - VTO Gradbeništvo: DELTA 400
- Tehniška fakulteta - VTO Elektrotehnika, računalništvo in informatika: DELTA 340
- Visoka ekonomsko komercialna šola: DELTA 644
- Visoka šola za organizacijo dela Kranj: DELTA 340

Fizične povezave med računalniki niso pomembne za uporabnike, zato so organizirane tako, da smo kar se da optimalno uporabili komunikacijske vmesnike.

Našo mrežo smo povezali tudi z računalnikoma drugačne arhitekture. Na Univerzi Edvarda Kardelja v Ljubljani uporabljajo računalnik DEC-10, ki je resda tudi Digitalov izdelek, vendar njegov operacijski sistem do nedavnega še ni podpiral DECNETa. Zato smo skupaj z ljubljanskimi kolegi poskrbeli za drugačno programsko opremo za povezavo. Na tej relaciji smo pravzaprav imeli dva različna softwara, namreč VAXNET in pa v zadnjem času vse popularnejši KERMIT. Od julija letos imajo tudi na DEC-10 DECNET, tako da smo zdaj tudi z njimi povezani na enak način, kot z drugimi centri naše Univerze.

Računalnik povsem drugega tipa pa je CYBER 72/172 na Republiškem računskem centru v Ljubljani. Z njim smo povezani preko emulatorja MUX 200, obstaja pa tudi povezava DEC-10-CYBER.

Težave pri izgradnji računalniške mreže

Na tem področju smo pri nas praktično orali ledino. Rešiti smo morali ogromno problemov, tako na hardwerskem kot na softwarskem področju. Kar se tiče ustrezne programske opreme, je bila zadeva vsaj med centri naše Univerze lažja, saj je obstajal Digital DECNET. Dosti težje je bilo z ustrezno komunikacijsko opremo. Koliko muke je bilo potrebno, da smo zbrali vse modeme, multiplekserje, DL-je, DUP-e, DMR-je in vse ostalo, kar k temu sodi. Poseben problem so predstavljale tudi razdalje med

posameznimi računalniki. Niti dva namreč nista v isti stavbi, kaj šele prostoru. Razdalje se gibljejo od 50 m pa vse do 200 km. Zaradi tega tudi hitrosti prenosa podatkov variirajo od 1200 bitov na sekundo do 56000 bitov na sekundo.

Na hude težave pa smo naleteli že pri samih pripravah na izgradnjo mreže. Na številnih seminarjih, srečanjih, tečajih in neposrednih pogovorih z raznimi strokovnjaki smo slišali ogromno o teoriji računalniških mrež, o tistih slavnih sedmih nivojih, o entropiji, o formulah za preračunavanje marsičesa, spoznali cel kup grafikonov in se naučili še dosti drugega. Zelo težko pa smo prišli do informacij, kako konkretno postaviti mrežo na noge, (čeprav je ustrezna programska oprema že sestavni del operacijskega sistema), sploh pa, kaj še vse pri tem potrebujemo. Dosti dela je bilo potrebno, da smo se premaknili od zagotovil proizvajalcev, da "mreža ni noben problem - to mora delati", bližje našemu cilju. Ob že znanih "kvalitetah" našega telefonskega omrežja so nas najbolj mučile nabave komunikacijskih vmesnikov, potrebnih za povezavo med računalniki. Vse navedene probleme smo dokaj uspešno rešili in mreža je končno zaživela.

Kaj smo pravzaprav s tako mrežo dosegli?

Denimo, da delamo z nekim računalnikom v naši mreži. Ne glede na naravo dela slej ko prej naletimo na določene težave, kot na primer:

- preskromna kapaciteta procesorja, kar pomeni daljši izvajalni čas s povečanim številom operacij;
- prezasedenost sistema v konicah;
- pri zahtevnejših programih pride do zasičenosti kapacitete centralnega in perifernega spomina;
- premajhna skupna kapaciteta diskovnih enot ne dopušča kreiranja večjih datotek ali podakovnih baz;
- okvara samo dela sistema (npr. edinega tiskalnika) nas tudi do 100 % onemogoči;
- podatke je treba med računalniki fizično prenašati (trakovi, nekompatibilnost medijev, ...);
- nimamo specialne opreme (grafika, posebni programi ...), ki jo ima na primer naš kolega v sosednjem centru;
- itd.

V mreži, kot je naša, se lahko s "svojega" računalnika s preprostimi ukazi enostavno "preselim" na nek drug računalnik in tam delam na enak način, kot da bi bil priključen direktno nanj, poleg tega pa lahko še prenašam podatke iz enega računalnika na drugi oziroma uporabljam podatke in programe iz drugih računalnikov. Vse, kar moram poznati, so ustrezne šifre oziroma gesla na drugem računalniku in pa seveda pravila njegovega operacijskega sistema, če gre za drug tip računalnika.

V približno dveh letih, kar dopolnjujemo in izgrajujemo našo računalniško mrežo, smo ugotovili najrazličnejše primere njene uporabe. Posamezni uporabniki na poljubnem računalniku se preko mreže selijo s sistema na sistem in na ta način obidejo vse težave, opisane v prejšnjem odstavku. Ugotovljen je ogromen prihranek na času pri prenosu podatkov med računalniki. Vsa zamudna dela s trakovi so odpadla. Nemalokrat smo bili v tak prenos dobesedno prisiljeni, saj je na primer odpovedala edina tračna enota na nekem računalniku, pa smo preko mreže spravili podatke na trak na drugem računalniku. Knjižnica Tehniške fakultete vzdržuje na Republiškem računskem centru podatkovno zbirko za INDOK za področje tekstila za Slovenijo. Podatki se zbirajo na centralnem univerzitetnem računalniku, potem pa se preko mreže pošljejo v RRC. Tako so podatki za vso Slovenijo na voljo takoj, medtem ko so bili prej stari dva do tri mesece, preden so po pošti prišli v Ljubljano s trakovi. Ko na primer dobi Knjižnica Tehniške fakultete zahtevo po iskranju nekih informacij iz podatkovne zbirke knjižničarka kar na terminalu na našem sistemu sestavi ustrezni paket ukazov, ki jih pošljemo preko mreže v izvajanje na Republiški računski center in potem čez čas dobi rezultate na terminal ali na papir.

Noben problem ni uporabljati različnih programskih produktov, kot so matematične knjižnice, konstrukcijski paketi, statistični in simulacijski programi itd., ki jih posamezni centri imajo in dajo na razpolago tudi drugim uporabnikom preko mreže. Zelo težko bi si vsak računalniški center na naši univerzi privoščil drage znane programske pakete, kot so NAG, SPSS, GPSS, IMSL, FINEL, BERSAFE in podobno. Tako dejansko nabavi vsak samo nekatere, potem pa jih lahko s pomočjo mreže uporablja tudi na drugih sistemih.

Senčne plati

V prejšnjem odstavku smo našteli nekatere pozitivne plati na naši mreži, ki so se pokazale v času njenega življenja. Seveda pa ni vse tako rožnato. Izkazalo se je, da je hitrost prenosa 1200 bitov na sekundo razmeroma skromna. Zares omembe vredni rezultati so edino na tisti relaciji, kjer je hitrost prenosa 56000 bitov na sekundo, čeprav so tudi pri 9600 in 4800 še zadovoljivi. Za distribuirano obdelavo, ki v določenih časovnih obdobjih pošilja podatke na druge računalnike (ali pa jih od tam uporablja), je taka mreža kar uspešna. Ni pa si mogoče predstavljati, da bi lahko v primeru okvare nekega računalnika vse njegove uporabnike ali pa vsaj večino, v najkrajšem času preselili drugam, z vsemi njegovimi podatki in programi vred.

Poleg tega so tu še tudi druge težave na manjših računalnikih, predvsem DELTA 340, pa tudi DELTA 644. Pri nekaterih majhnih konfiguracijah lahko DECNET močno obremeni računalnik, gre pa tudi za nekompatibilnost sistemske programske opreme, saj ponekod ni možno hkrati izvajati DECNET-a in programa za beleženje porabe sistemskih resursov (accounting). Upamo, da bomo to rešili s povečanjem centralnega pomnilnika na nekaterih sistemih in s poenotenjem sistemske programske opreme na sorodnih računalnikih.

Ne nazadnje je treba spregovoriti tudi o varnosti in zaščiti podatkov. V mreži, kot je naša, ne obstaja poseben zaščitni mehanizem proti uporabnikom z drugih računalnikov, kar seveda dela dodatne preglavice odgovornim delavcem pri posameznih računalnikih. Res pa je, da zato tudi ni odnosa gospodar-suženj, ampak so vsi računalniki v mreži enakopravni.

S tem referatom smo skušali prikazati nekatere naše izkušnje z računalniško mrežo. Sodimo, da smo na tem področju prišli dokaj daleč v našem okolju, kar pa seveda ne pomeni konca, ampak še večjo obvezo za iskanje novih poti za širjenje in razvoj računalniških mrež. V najkrajšem času bomo povezali vse tri računalnike DELTA 4850 v Računalniškem centru Univerze v Mariboru. Če ne bomo uspeli nabaviti komunikacijskih vmesnikov za DECNET, bo za silo zadostoval tudi skoraj vsepovezujoči KERMIT. Kasneje bomo te tri računalnike, pa tudi še 2-3, ki so relativno blizu, najverjetneje povezali v mrežo ETHERNET, seveda pa je še privlačna varianta clusterjev. Sedaj načrtujemo povezavo s še nekaterimi računalniki, tudi na republiškem nivoju, s čimer bi olajšali dostop do po-

trebnih podatkov, ki se zbirajo na enem mestu, (na primer Republiški zavod za statistiko). Tu gre tudi za distribuirane baze podatkov na širšem področju, na primer jugoslovanskem ali evropskem ter za vključevanje v JUPAK-jugoslovansko PTT omrežje za prenos podatkov.

REALIZACIJA PROTOKOLA X.25 - LAPB ZA DIGITALNO NAROČNIŠKO ZANKO

ISKRA TELEMATIKA KRANJ
TOZD RAZISKAVE IN RAZVOJ
Bizjak Igor, Čepon Stanislav, Györköš Daniel, Kunčič Sašo

UDK: 681.3.007

Digitalizacija telefonskega omrežja vodi do vpeljave ISDN (integrated service digital network). Tudi ISKRA Telematika se je vključila v razvoj komutacijskih sistemov na tem področju. Eksperimentalni teleinformatski sistem (ETS) predstavlja prvi korak v ISDN in je rezultat lastnega znanja. Realizacija protokola X.25 zavzema pomembno mesto v tej aplikaciji.

Digitization of telephone network postulated for the introduction of ISDN(integrated service digital network).Within this concept, ISKRA Telematika has started to develop such switching systems. ETS experimental teleinformation system represents the first step in ISDN and is the result of our own knowledge. Realisation of X.25 protocol plays on important part within this application.

1. Uvod

ETS je nadaljevanje razvoja na digitalnem telefonskem aparatu (DTA) [1] in digitalni naročniški zanki (DNZ) [2]. Sistem obsega konfiguracijo 6 DTA in 2 standardna telefonska aparata (analogna). Sl. 1.1.prikazuje konfiguracijo ETS-a.

Osnovne lastnosti:

a) neodvisno krmiljenje govornih in podatkovnih funkcij;
b) vspostavljanje - rušenje podatkovne zveze avtomatsko v podatkovnem terminalu - računalniku. (PT-R) s protokolom V.25 ali ročno preko DTA;
c) aplikacija podatkovne zveze je določena s tipom PT-R in aplikacijo programske opreme;
d) lokalna govorna zveza;
e) lastnosti digitalne naročniške zanke (DNZ)
   - dvosmerni dvožični prenos po navadni telefonski parici,
   - hitrost prenosa informacije je 144 kb/s,
   - organizacija kanala B + B + D
     B kanal - 64 kb/s prenos govora (PCM)
     B kanal - 64 kb/s prenos podatkov (V.25, RS 232)
     D kanal - 16 kb/s prenos signalizacije (X.25 - LAPB)
   - napajanje DTA iz centrale (uporabljena CMOS tehnologija),
   - nivojska in modularna zgradba programske in materialne opreme.

Sl. 1.2. prikazuje funkcijski model DNZ s stališča ISO (Internal organization for standardization) modela. Digitalni naročniški modul (DSM) v tej fazi še ni sposoben obdelave DTA (predvsem zaradi signalizacije) je bilo razvito digitalno naročniško vezje (DNV).

Nivojska zgradba pomeni, da npr. n-ti nivo za svoje delovanje koristi usluge (n-1) nivoja in daje usluge (n+1) nivoju.

Naloge nivojev DNZ:

Nivo 1: - realizacija digitalnega prenosa po bitih
- časovna ločitev smeri
- hitrost prenosa 160 kb/s
- dodajanje - izdvajanje napajanja na linijo

Nivo 2: - obdelava govorne informacije
- podatkovni vmesnik za priključitev terminala (V.25, RS 232)
- realizacija protokola V.25
- realizacija protokola X.25 - LAPB

Nivo 3: - v DTA
- nadzor tastature
- izpis numeričnih podatkov na LCD prikazovalnik
- realizacija zvočnega alarma
- v DNV
- realizacija komunikacije na signalizacijskem vodilu v DSM
- transformacija ukazov med DTA in DSM

Vsa komunikacija med DTA in DNV poteka po D kanalu za potrebe DTA in protokola V.25.

2. Implementacija protokola X.25 - LAPB

Protokol X.25-LAPB predstavlja linijski nivo (nivo 2) po ISO modelu in opisuje procedure za izmenjavo podatkov med DCE (Data Circuit Terminating Equipment) in DTE (Data Terminal Equipment). V ETS predstavlja DTE digitalni telefonski aparat (DTA) in DCE digitalno naročniško vezje (DNV) - skupaj z digitalnim naročniškim modulom (DSM). LAPB (Link Access Procedure Balanced) je zasnovana na osnovi HDLC (High-level Data Link Control) protokola. Protokol v bistvu predstavlja niz pravil, ki koordinirajo in zagotavljajo prenos podatkov. Specifikacija protokola je zasnovana iz treh delov:
- sintakse: definira elemente procedure,
- semantika: definira izmenjavo uslug nivojev,
- časovne razmere: definira maksimalno zakasnitev pričakovanega odgovora in izpada nižjega nivoja.

Osnovna oblika prenosa sporočil je okvir (frame). Prenos okvirjev se vrši v modulih.

CCITT priporočila za protokol X.25-LAPB zelo sistematično opisuje razne ukaze in načine prenosa v raznih situacijah, vendar na račun funkcionalne zgradbe. Zelo težko si je ustvariti globalno sliko zaradi takšnega opisa.

Vsiljuje se ideja o realizaciji LAPB procedure s pomočjo končnega avtomata stanj. Izkazalo se je, da je mogoče zelo enostavno in pregledno zasnovati in realizirati programski model. Končni avtomat mora realizirati akcije kot so: vspostavljanje in rušenje zveze, vzdrževanje zveze, če ni podatkov za prenos, detekcijo napake, ponavljanje, itd. Te akcije so v modelu končnega avtomata predstavljene kot funkcije prehoda med stanji. Model končnega avtomata je realiziran programsko.

Po analizi vseh akcij je bil končni avtomat razdeljen na tri globalna stanja:
a) postavljanje in rušenje zveze,
b) izmenjavo sporočil,
c) reset zveze in nastop semantične napake.

Vsako globalno stanje je potem še razdeljeno na podstanja, ki so osnovna stanja modela.

3. Zgradba programske opreme

Celotna programska oprema (PO) je zasnovana na principih modularnosti, posamezni moduli pa so realizirani z modelom končnega avtomata stanj.

V sl. 2.1. na strani DTA imamo tri časovno neodvisne module (PO DTA, PO V.25, PO LAPB). Časovna neodvisnost modulov pogojuje takšno zgradbo programskega modula, da je procesno moč (µR) zasedena samo toliko časa, kolikor je nujno potrebno. Potem se procesna moč preda drugemu modulu. Koncept končnega avtomata podpira takšno zgradbo, saj lahko prekinemo izvajanje in nadaljujemo ponovno v istem stanju. Moduli ne smejo vsebovati zank (zakasnitev - delay). Model LAPB procedure prikazuje sl. 2.2.. Izvajalec poskrbi, da se izvajanje nadaljuje v stanju, kjer je bilo prekinjeno in izvrši prehod v novo stanje, ki je bilo definirano z določeno akcijo. Posamezno stanje končnega avtomata je realizirano v obliki pogojnih stavkov, kar omogoča enostavno testiranje programa. Sprejem in oddaja sta asinhrona in podprograma iz sl. 2.2. preverita sprejeti oziroma formirata oddani okvir glede na semantiko in sintakso.

Celotna programska oprema je strukturno zgrajena in napisana v zbirniku za µP MC 146805.

4. Zaključek

Princip uporabe končnega avtomata je pokazal sledeče prednosti:
a) podpira modularnost,
b) strukturna zgradba in preglednost realizacije končnega avtomata,
c) enostavno testiranje programske opreme,
d) zanesljivost,
e) podpira časovno neodvisnost modulov.

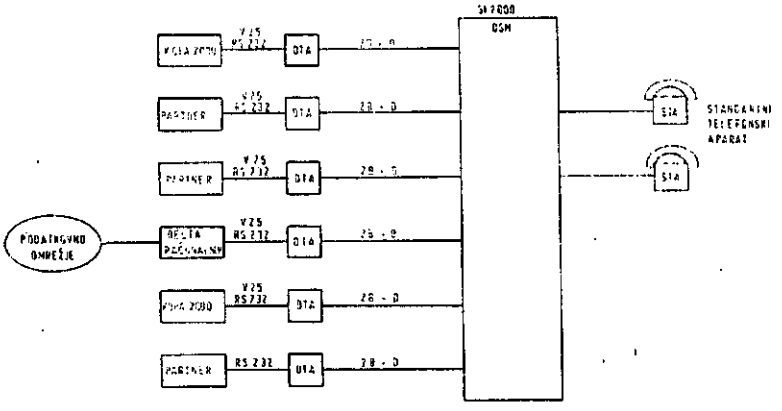
Testiranje PO je potrdilo upravičenost takšnega pristopa pri realizaciji programske opreme. Spreminjanje končnega avtomata je enostavno spreminjanje pogojnih stavkov, ki tvorijo program stanja.

5. Reference

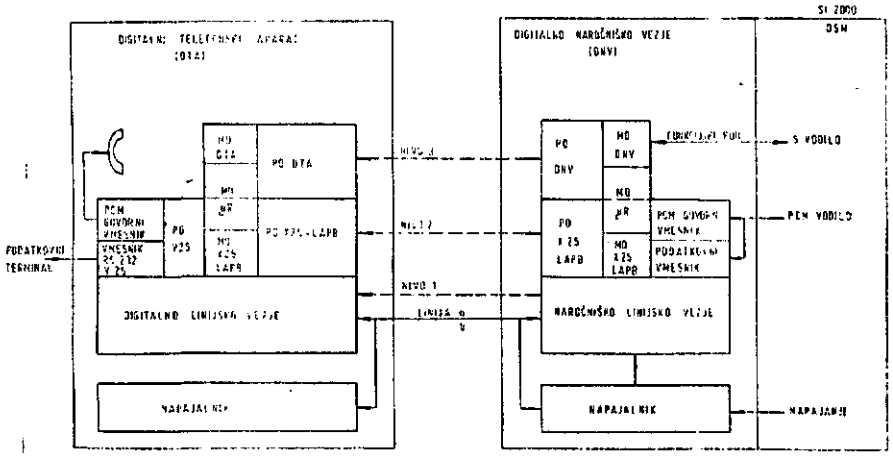
1 YUTEL 80/III E-2 Komunikacijski terminali, ki jih omogoča digitalni telefonski kanal.

2 YUTEL 84/III G-10 Prenos informacije po digitalni naročniški zanki.

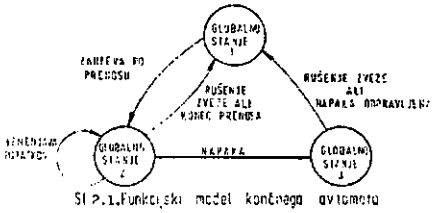
3 CCITT Recommendation X.25 VIII.2.



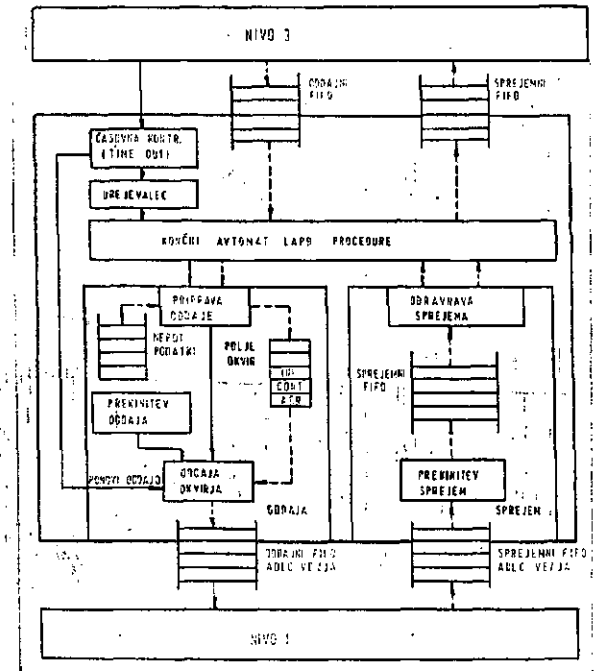
Sl 11 Funkcijski model ETS



Sl 12 Funkcijski model DNZ s stališča ISO modela



Sl 2.1.Funkcijski model končnega avtomata



Slika 2.2: Programski model LAPB procedur

MULTIMIKRORAČUNALNIŠKE MREŽE KOT ALTERNATIVA ZA MINIRAČUNALNIKE V KOMPLEKSNIH SISTEMIH VODENJA PROCESOV V REALNEM ČASU

Borut Črnivec, Iskra - Avtomatika, ZOZD Sistemi, Ljubljana

UDK: 681.3.007

V referatu je opisana konkretna izvedba centra operativnega vodenja napajalnega omrežja železnice na progi Sofija - Plovdiv z lokalno mikroračunalniško mrežo ZI-30. Narejena je analiza in primerjava izbrane decentralizirane realizacije s centralizirano realizacijo, v obliki enega miniračunalnika. Razčlenjena je zmogljivost, kapaciteta, zanesljivost in ekonomičnost obeh variant.

MULTIMICROCOMPUTER NETWORK AS AN ALTERNATIVE FOR MINICOMPUTERS WITHIN COMPLEX REAL TIME PROCESS CONTROL SYSTEMS. In this article the local microcomputer network ZI-30, used in the power control centre for the railway Sofia - Plovdiv, is described. The chosen decentralized realization is analysed and compared with a centralized realization involving one minicomputer. The aspects of performance, capacity, reliability and cost-effectivness are considered.

1. UVOD

Razvoj sodobne mikroračunalniške tehnologije omogoča realizacijo decentraliziranih in distribuiranih sistemov vodenja procesov. Prostorska distribuiranost (informacije se procesirajo na mestu nastanka) je logična posledica distribuiranosti vodenega procesa, saj je taka realizacija ponavadi najcenejša. Kompleksne naloge in funkcije, ki se lokacijsko izvajajo na enem mestu in so bile običajno realizirane z miniračunalnikom, pa lahko funkcijsko distribuiramo - smiselno porazdelimo naloge vodenja na posamezne mikroračunalniške podsisteme, povezane v multimikroračunalniški sistem (mrežo).

Za učinkovito realizacijo funkcijsko distribuiranega sistema vodenja moramo zadostiti vrsti pogojev in kriterijev. Najpomembnejše zahteve so:

- maksimalno možna avtonomija podsistemov,
- minimalni pretok informacij med podsistemi v mreži,
- časovno optimalen in zmogljiv komunikacijski protokol za izmenjavo informacij med podsistemi,
- možnost centralne diagnostike in nadzora mreže,
- velika fleksibilnost pri izbiri konfiguracije (opreme in funkcij),
- kvalitetna programska oprema (modularnost, transparentnost, fleksibilnost).

Ob zgornjih predpostavkah pa distribuiran sistem nudi vrsto prednosti v času projektiranja, programiranja in obratovanja:

- večja preglednost in razumljivost,
- poenostavljeno projektiranje,
- olajšana je etapna izgradnja sistema,
- visoka modularnost,
- poenostavljeno testiranje,
- visoka zanesljivost,
- visoka razpoložljivost.

2. LOKALNA MIKRORAČUNALNIŠKA MREŽA ZI-30

Tipičen primer konkretne aplikacije predstavlja operativni center vodenja napajalnega omrežja železnice na progi Sofija-Plovdiv.

Distribuiran sistem vodenja je realiziran z lokalno mikroračunalniško mrežo ZI-30. Le-ta obsega do 64 mikroračunalniških podsistemov (v konkretnem primeru 10), ki so med seboj povezani s serijskimi vodili (SVOD). Dolžina vodil je lahko do 1000m pri hitrosti prenosa 50kBd. Posamezni funkcijski podsistem je lahko sestavljen iz enega ali več mikroračunalnikov (glavni in do pet podrejenih), ki so med seboj hierarnično povezani s paralelnimi vodili (VVOD). Dolžina paralelnih vodil je do 7,5m pri hitrosti prenosa 1,2M 8-bitnih besed/s. Vsi mikroračunalniki so osnovani na mikroprocesorski družini Motorola M6800. Zaradi povečanja zanesljivosti lahko posamezne podsisteme (delno ali v celoti) podvojimo.

Serijska sistemska vodila, ki predstavljajo hrbtenico multimikroračunalniškega sistema, so zaradi zanesljivosti podvojena (SVOD/A, SVOD/B). Komuniciranje vodi arbiter (eden izmed mikroračunalnikov), ki izvrši avtomatski preklop iz enih na druga vodila, ko odkrije napako pri komunikaciji. Zaradi zanesljivosti ima funkcijo arbitra več mikroračunalnikov, od katerih je eden aktiven, ostali so pasivni arbitri. Aktivni arbiter ciklično dodeljuje vodila posameznim podsistemom, ti pa neposredno komunicirajo med seboj. Sporočilo obsega organizacijski, podatkovni in zaščitni del. Prvi vsebuje ciljno adreso in krmilne signale, drugi obsega lo 255 8-bitnih besed, zaščito pa predstavlja geometrijski kod. Ciljna adresa je indeks logičnega kanala ali signala, ki ustreza je v splošnem fizični naslov v konkretni mikroračunalnik mreže transparentno.

3. DISTRIBUCIJA FUNKCIJ

Center nadzoruje in vodi napajalni sistem za 200 km proge. Neposredni zajem informacij in izdajo komand v proces izvaja skupaj 27 daljinskih (končnih) postaj. Skupno število informacij v procesu znaša cca 1000 komand (dvobitni izhodi), 12 nastavitvenih vrednosti (5 ali 6 bitni izhodi), 650 enobitnih in 500 dvobitnih signalizacij (vhodi), 66 analognih meritev, 12 števčnih vrednosti. Naloge centra funkcijsko grupiramo v pet razredov, dejansko pa razdelimo med deset mikroračunalniških podsistemov:

PREDPROCESIRANJE INFORMACIJ. Pet podsistemov predstavlja čelne predprocesorje (PP), ki komunicirajo s končnimi postajami (preko VZU zunalov), predstavljajo stanje procesa na sinoptični (mozaik) plošči in numeričnih prikazovalnikih. V njih je implementirana distribuirana podatkovna baza s pripadajočimi internimi in eksternimi funkcijami, kjer se izvede osnovna obdelava informacij iz končnih postaj (ugotavljanje sprememb signalizacij in njihov prikaz na sinoptiki, izračun izrabe odklopnikov, izračun 15-minutnih povprečij merilnih vrednosti, vodenje dnevne porabe energije, nadzor prekoračenja mej merilnih vrednosti ipd.). Trije predprocesorji so s pripadajočimi (tremi) končnimi postajami povezani na način točka-točka, ostala dva pa imata vsak svojo grupo (A in B) končnih postaj povezanih na linijski način (multipoint). Taka konfiguracija je posledica različne pomembnosti in obsega informacij v določenih končnih postajah.

OBDELAVA. Podsistem ORD je namenjen specifičnim obdelavam (izračuni porabe aktivne in reaktivne energije ter cos(fi) po različnih odvodih za različna časovna obdobja v različnih tarifnih območjih, izračun lokacije kratkega stika ipd.). Temu podsistemu je (kot tudi vsem petim predprocesorjem) zaradi matematičnih obdelav dodan aritmetični procesor (floating point processor Am9511A), ki deluje paralelno mikroračunalniku in močno poveča zmogljivost podsistema.

VODENJE PROCESA. Dva podsistema (POSL A in B) sta namenjena operaterjevim posegom v sistem (posluževanje). Operater izdaja komande v proces (krmiljenje ločilk, trafo stopenj, ventilatorjev, itd.) in izbira oziroma zahteva prikaz različnih informacij stanja procesa v različnih oblikah (spremljanje merilnih vrednosti, izpis merilnih protokolov, izpis arhiviranega protokola, osvetlitev, zatemnitev in test posameznih delov sinoptike, kvitiranje alarmnih signalizacij ipd.). Opisane naloge so temeljnega pomena za spremljanje in vodenje procesa, zato jih izvaja le eden od obeh podsistemov, drugi pa prevzeg spremlja v stanju aktivne rezerve (stand-by). S tem močno povečamo zanesljivost, kot bo razvidno iz analize v nadaljevanju članka.

PROTOKOLIRANJE. Podsistem PROT je namenjen spremljanju procesa z obratovalnim protokolom. Tako dokumentiramo vse dogodke v vodenem procesu, vse posege operaterja v proces oziroma sistem, kot tudi dogodke v sami mikroračunalniški mreži centra, kar je posebej pomembno za hitro odkrivanje in in uspešno odpravljanje napak (diagnostika).

ARHIVIRANJE. Podsistem ARH krmili diskovno enoto, na katero arhiviramo vrsto procesnih spremenljivk kot so povprečne vrednosti meritev napetosti, toka, delovne in jalove moči, ure in dnevna poraba električne energije po posameznih enotah, izraba odklopnikov ipd. Zaradi pomembnosti arhiviramo tudi obratovalni protokol tekočega in predhodnega dneva. Poleg arhiviranja podsistem tudi neposredno izdaja arhivirane informacije v obliki različnih zaprtih protokolov (tabele in grafi vrednosti meritev in porabe električne energije, arhivirani odprti protokol) na pisalnik.

4. ANALIZA IZBRANE ALTERNATIVE

Opisani distribuirani multimikroračunalniški sistem bomo primerjali z enoračunalniško izvedbo. Za primerjavo vzemimo miniračunalnik iz družine PDP-11 (npr. 11/34), klasičen procesni računalnik, ki pa je še danes priseten v mnogih aplikacijah.

4.1. Zmogljivost

Pri analizi je bistveno upoštevati specifičnost nalog v centru vodenja pri posamezni aplikaciji. V opisanem primeru prevladujejo (cca 70%) logične operacije (postavljanje, brisanje bitov v podatkovnih strukturah, krmiljenje bitnih izhodov) in operacije urejanja, iskanja in prepisovanja, pri katerih je miniračunalnik 11/34 približno 6 do 8 krat hitrejši od mikroprocesorja MC 6800. V distribuiranem centru paralelno deluje 9 podsistemov (eden je v aktivni rezervi), pri čemer ob minimalni komunikaciji po mreži ne čutimo distribuiranosti, ker ima vsak podsistem vsaj dva procesorja (eden komunicira z mrežo, drugi opravlja aktivne funkcije). V primeru aritmetičnih operacij je prednost na strani miniračunalnika jasna in težko primerljiva. V našem primeru je aritmetičnih operacij sorazmerno malo in so dokaj preproste, tako da postanejo mikroračunalniki ob dodatku aritmetičnega procesorja primerljivi z miniračunalnikom. Opisana izvedba predstavlja torej operativno vodenje procesa v realnem času. Če zahtevamo funkcije podaljšanega realnega časa, ki zahtevajo množico kompleksnih aritmetičnih operacij in obdelav, katerih rezultate vračamo v proces (npr. z optimiranjem regulacijskih algoritmov) v cilju povečanja zanesljivosti, ekonomičnosti ipd., je miniračunalnik primernejši in sposobnejši. Pri opisanih funkcijah je zmogljivost obeh realizacij približno enaka, oziroma ostaja rahla prednost na strani multimikroračunalniške mreže.

Vsi programi so fiksno shranjeni v hitrem pomnilniku (EPROMu), zato odpade čas nalaganja programov z diskovne enote v pomnilnik, ki je pri miniračunalniku prisoten. Operacijski sistem je distribuiran, zato je čas dodeljevanja CPE ustrezno krajši, kar povečuje zmogljivost.

4.2. Kapaciteta

Kapaciteta hitrega pomnilnika. Obe alternativi nudita mnogo večjo kapaciteto od potrebne. V distribuiranem sistemu lahko vsak mikroračunalniški podsistem direktno naslavlja 64K, po potrebi, ob naslavljanju po straneh pa do 256K 8-bitnih besed. Devet podsistemov bi lahko imelo preko 2M besed pomnilnika, za opisano realizacijo pa predvidevamo 450K besed. Miniračunalnik 11/34 direktno naslavlja 32K, ob razširjenem naslavljanju pa tipično 128K, oziroma maksimalno 2M 16-bitnih besed ob ustrezni aparaturni opremi.

Kapaciteta diskovne enote. V opisani aplikaciji na diskovne enote shranjujemo [illegible]

procesnih informacij je za kratko časovno obdobje (urni in dnevni arhiv) in relativno malo informacij za daljše obdobje (mesečni in letni arhiv). Tako zasedemo le cca. 10% celotne kapacitete diskovne enote sistema TI-30, ki znaša 5 Mbytov. V primeru bistveno večjega obsega informacij, ki se tipično pojavi pri funkcijah v podaljšanem realnem času, je očitna prednost na strani miniračunalnika, ki mu enostavno izberemo ustrezni diskovni pogon, praktično poljubne kapacitete (npr. 10, 20, 40, celo 160 ali 300 Mbytov) glede na potrebe dane aplikacije.

4.3. Zanesljivost in razpoložljivost

Za oceno dejanske zanesljivosti je potrebno razpolagati s statističnimi podatki o obratovanju in okvarah že delujočih sistemov. Take podatke je od proizvajalcev opreme za procesno vodenje težko dobiti, ker predstavljajo vrsto poslovne tajnosti. V primeru sistema TI-30 razpolagamo s teoretično izračunano zanesljivostjo aparaturnih modulov. Izračun je narejen na podlagi zanesljivosti vgrajenih elektronskih komponent (integriranih vezij) posameznih proizvajalcev. zanesljivost vsakega podsistema mreže je izračunana na osnovi uporabljenih modulov. Kljub precejšnim naporom ni bilo mogoče dobiti ekvivalentnih podatkov za miniračunalnik, zato bo primerjava le načelna ob sicer utemljeni predpostavki, da je zanesljivost aparaturne opreme miniračunalnika približno enaka zanesljivosti ustreznega mikroračunalniškega podsistema, saj sta oba zgrajena iz podobnih elementarnih komponent svetovnih proizvajalcev. Zgornja predpostavka še posebej velja za pomnilniška vezja, zanesljivost hitrega pomnilnika pa je temeljni faktor v zanesljivosti celote, ker je bistveno manjša od zanesljivosti ostalih gradnikov (procesorja, dajalnika časovnih taktov ipd.). Zanesljivost lahko predstavimo s povprečnim časom med izpadi (MTBF), večja vrednost MTBF pomeni torej večjo zanesljivost. Za primerjavo obeh alternativ uvedemo spremenljivko k, ki naj predstavlja kvocient: k= (MTBF multimikroračunalniške izvedbe) / (MTBF miniračunalniške izvedbe). Pri k>1 je prednost na strani mreže, pri k<1 na strani miniračunalnika. Ob interpretaciji rezultatov je nujno upoštevati cilj, ki smo si ga zadali pri realizaciji centra vodenja - zagotoviti visoko zanesljivost najpomembnejših funkcij: izdaje komand v proces (posegi operaterja) in spremljanje procesa (obratovalni protokol in odsev stanja procesa na sinoptični plošči). S podvojitvijo podsistema za posluževanje in z arhiviranjem obratovalnega protokola (podvojitev) povečamo zanesljivost za faktor 3,6 oz. 3,2 glede na nepodvojeno izvedbo. Sledijo rezultati (kvocient k) za nekaj zanimivih nalog oz. funkcij:

- popolno delovanje celotnega sistema (0.64),
- popolno delovanje vseh funkcij razen prikaza na sinoptični plošči (0.44),
- arhiviranje informacij iz vseh končnih postaj (0.5),
- arhiviranje informacij iz grupe končnih postaj (1.6),
- izpis arhiviranih podatkov (2.7),
- protokoliranje dogodkov iz vseh končnih postaj (0.7),
- protokoliranje dogodkov iz grupe končnih postaj (36),
- izdaja komand v vse končne postaje (0.7),
- izdaja komand v grupo končnih postaj (38),
- prikaz informacij iz vseh končnih postaj na sinoptični plošči (0.95),
- prikaz informacij iz grupe končnih postaj na sinoptični plošči (15000).

Kljub temu, da je verjetnost pravilnega delovanja vseh podsistemov v mreži bistveno manjša od verjetnosti delovanja pripadajočega miniračunalnika (k=0.64, več procesorjev, večja verjetnost okvare), je zanesljivost zantevanih bazičnih funkcij mnogo večja kot pri miniračunalniku. MTBF do izpada popolnoma vseh funkcij pa je 6 milijonkrat daljši kot pri enoračunalniškem sistemu!

Podobno kot zanesljivost je bistvenega pomena razpoložljivost sistema (r), ki jo pri konstantni intenzivnosti odpovedi definiramo kot kvocient r=MTBF/(MTBF+MTTR), kjer MTTR predstavlja srednji čas do popravila sistema. MTTR lažje minimiziramo v mikroračunalniški mreži, kjer hitreje lociramo okvare aparaturne in programske opreme (oboje je distribuirano). Če imamo v rezervi cel nabor aparaturnih modulov (kar ni bistvena postavka v primerjavi s celotnim številom modulov - v našem primeru 11 proti 131), lahko te okvare praktično takoj odpravimo in dosežemo izredno visoko razpoložljivost, saj je MTTR zanemarljiv v primerjavi z MTBF, ki je za diskovno enoto tipično 8000 ur, za ostale gradnike pa več let.

4.4 Cena

Cena obeh alternativ je približno istega reda. Cena namišljene variante je ocenjena na podlagi funkcijsko podobnih projektov, ki jih realiziramo z miniračunalniki.

5. ZAKLJUČEK

Na podlagi analize ugotovimo, da je multimikroračunalniška mreža v primeru opisane aplikacije adekvatno in uspešno nadomestilo za miniračunalnik, s tem da v vrsti pogledov nudi očitne prednosti. Najpomembnejše je bistveno povečanje zanesljivosti in razpoložljivosti.

prednost in ekonomičnost distribuiranega procesiranja se kaže ravno pri povečevanju zanesljivosti samo določenih funkcij (podsistemov), ne pa pri večanju zanesljivosti vseh funkcij (podvajanje celotnega sistema).

V primeru bistveno obsežnejših funkcij in obdelav (npr. podaljšan realni čas) je potrebno uporabiti miniračunalnik. Idealno rešitev za take probleme predstavlja vključitev ustreznega miniračunalnika direktno v mikroračunalniško mrežo (na serijska vodila). Tako lahko ob veliki zmogljivosti ceneno povečujemo zanesljivost določenih funkcij (mikroračunalniških podsistemov).

LITERATURA

/1/ N.Panić, L.Lenart, P.Peterlin in drugi: Multimikroračunalniške izvedbe kompleksnih sistemov vodenja procesov v realnem času, Informatica 2/3, Ljubljana, 1983

/2/ Distributed Systems Handbook, Digital Equipment Corporation, 1978

/3/ R.H. Eckhouse, L.R. Morris: Minicomputer Systems; Organization, Programming and Applications (PDP-11), Prentice-Hall, New Jersey, 1979

/4/ Jernej Virant: Zanesljivost računalniških sistemov, Univerza v Ljubljani, Fakulteta za elektrotehniko, 1981

POVEČANJE INFORMACIJSKIH PRETOKOV S TRANZITNIM PRETVORNIKOM

B.Pehani, B.Pohar, J.Bešter
Fakulteta za elektrotehniko
Ljubljana, Jugoslavija

UDK: 681.3.007

POVZETEK - Referat opisuje funkcijo tranzitnega pretvornika, njegovo zasnovo in vključitev med obstoječe telefonsko in teleks omrežje. Uporaben pa je tudi okrnjen, direktno priključen med računalnik in teleks omrežje. Analizirane so različne stopnje njegove okrnjenosti in izpeljane njih karakteristike.

ABSTRACT - The paper describes the function of transit converter, its concept and its introduction between existing public telephone and telex networks. It is useful also truncated, connected directly between computer and telex network. Various degrees of truncation and their characteristics are analysed.

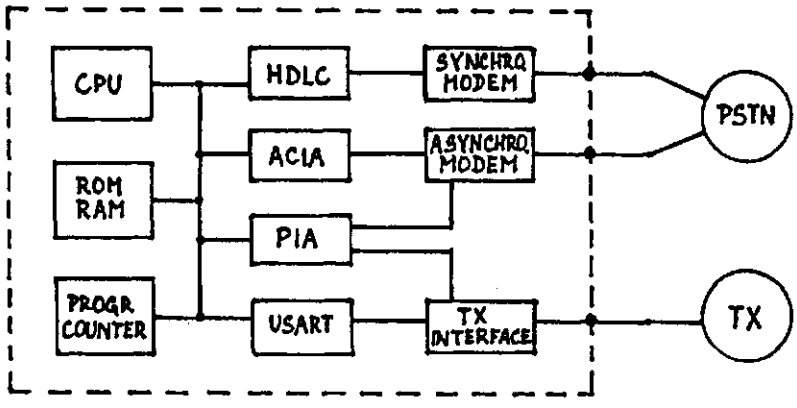
UVOD

Smisel osebnih računalnikov niso igre, ampak resnejše delo in njihova uporaba na vseh področjih aktivnosti našega življenja. To lahko dosežemo z vključitvijo osebnih računalnikov v hišne računalniško vodene sisteme in v splošni telekomunikacijski terminal uporabnika. Slednji naj bi omogočal poleg govornih komunikacij tudi podatkovne komunikacije v različnih oblikah. Hkrati s tem je potrebno na široko odpreti uporabo telekomunikacijskih omrežij uporabnikom prenosa podatkov. Tu pa se odpre vprašanje kateri prenosni medij je takoj, oziroma za male investicije na voljo v te namene. Vprašanje je še posebej pomembno za dežele z gospodarskimi težavami z omejenimi investicijskimi možnostmi. Na razpolago sta nedvomno obstoječe javno telegrafsko in telefonsko omrežje. Slednje nudi uporabniku tudi možnosti integriranega priključka za izmeničen telefonski in podatkovni prenos. **Podatkovni prenos** tega priključka pa lahko razširimo tudi na obstoječe telegrafsko omrežje. Tako lahko vsak računalnik, ki izpolnjuje določene pogoje, nastopa kot TX priključek in vsak TX priključek lahko nastopa kot asinhroni računalniški terminal. Potrebno je le povezati obe omrežji s tranzitnim pretvornikom (TP).

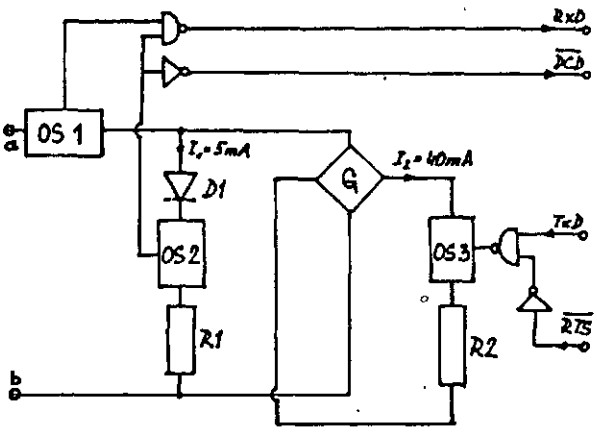
FUNKCIJA TRANZITNEGA PRETVORNIKA

Naloga tranzitnega pretvornika je, da omogoči in transformira prehod podatkov med telefonskim in teleks ali drugimi telekomunikacijskimi omrežji. Priključen je na priključke telefonskega in priključke teleks, oziroma drugih omrežij. Priključitev na teh točkah je najbolj standardizirana. Bolj podrobna razčlenitev funkcij tranzitnega pretvornika je naslednja:

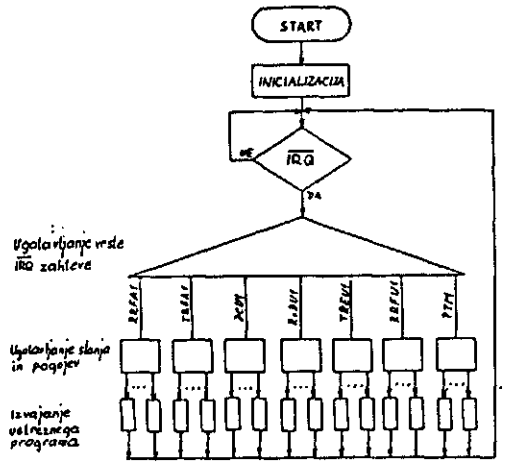
- vzpostavlja dohodne in odhodne telegrafske zveze,



Slika 1. Zgradba tranzitnega pretvornika



Slika 2. TX vmesnik



Slika 3. Diagram poteka delovanja TX

- vzpostavlja dohodne in odhodne telefonske zveze,
- preklopi telefonsko zvezo v podatkovno, prilagodi **hitrost** prenašanja podatkov uporabniku in oddaja uporabniku navodila,
- pretvarja znake in hitrost pri izmenjavi podatkov med teleprinterjem TX in računalnikom in krmili podatkovni pretok [1].

Povezava telegrafskega in teleks omrežja s tranzitnim pretvornikom prinaša naslednje koristi:

- vsak računalnik, ki izpolnjuje določene pogoje lahko nastopa kot TX priključek,
- vsak TX priključek lahko nastopa kot računalniški terminal; s pomočjo še prostih znakov (CCITT abeceda štev.2) ter tako omogočimo oddajo kompletnega nabora ASCII znakov, če je to potrebno.

ZGRADBA TRANZITNEGA PRETVORNIKA

Zgradba tranzitnega pretvornika je prikazana na sliki 1. Jedro predstavlja CPU (Motorola MC6800) in pripadajoči ROM/RAM. Ostali deli so periferne enote ACIA (MC6850), PIA (MC6821) in USART (Intel 8251A). Specifični periferni enoti pa sta HDLC vmesnik in TX vmesnik.

Vezje TX vmesnika je prikazano na sliki 2. Ta vmesnik opravlja naslednje funkcije:

- zaključuje naročniško telegrafsko zanko (priključka a in b), in določa tok I=5 mA z uporom R1 in tok I=40 mA z R2,
- razpoznava polariteto na žilah a in b ter jo sporoča vmesniku USART s signalom DCD (± polariteto pomeni logična enica),
- razpoznava prekinitve telegrafske zanke (telegrafski znaki) in jih sporoča vmesniku USART s signalom RxD,
- od USART sprejema signal RTS za prekinitev zanke, oziroma povečanje toka v zanki,
- od USART sprejema telegrafske znake preko signala TxD,
- električno ločuje telegrafsko omrežje in vmesnik USART in omogoča pretok podatkov ter signalizacije preko optosklopnikov OS1, OS2 in OS3.

Programsko opremo tranzitnega pretvornika sestavljajo:
- program inicializacije,
- program za ugotavljanje IRQ zahteve,
- programi, ki se vključujejo glede na IRQ zahtevo in stanje zveze (izvršijo prehod v novo stanje, oziroma nalogo potrebno ob IRQ zahtevi - prekodiranje kode, izpis navodila ...),
- tabele za prekodiranje ASCII v telegrafsko kodo in obratno,
- besedila, ki se izpisujejo kot navodila uporabniku.

Naslednja slika 3 prikazuje diagram poteka delovanja tranzitnega pretvornika. Zveze skozi tranzitni pretvornik se odvijajo s prehajanjem iz enega stanja v drugo. Prehode izvedejo ustrezni programi, katerih vključitev je odvisna od zunanjih in notranjih pogojev ter stanja zveze. Zunanji pogoji so IRQ zahteve, ki lahko pridejo iz naslednjih izvorov:

RRFA1 - sprejemni register ACIA je poln,
TREA1 - oddajni register ACIA je prazen,
DCU1 - sprememba polaritete TG priključka,
RxDU1 - prekinitev TG zanke za več kot 300 ms,
TREU1 - oddajni register USARTje prazen,
RRFU1 - sprejemni register USART je poln
PTM - prekinitev programabilnega števca (časovnega razzona)

Notranje pogoje (časovna kontrola,konec določenih akcij itd.) pa si program postavlja sam.

RAZLIČNE VARIANTE TP IN NJIHOVE KARAKTERISTIKE

Tranzitni pretvornik v različnih variantah lahko uporabljamo na različnih mestih:
- za povezavo računalnika, priključenega z modemom na javno komutirano omrežje (PSTN) s TX omrežjem (glej sliko 4),
- za povezavo računalnika direktno s TX omrežjem (glej sliko 5),
- za interne potrebe PTT, kot je to na primer avtomatska oddaja telefonsko sprejetih telegramov.

Najenostavnejši tranzitni pretvornik v sliki 5 je TX vmesnik prikazan na sliki 2. V tem primeru mora opravljati kompletni nadzor nad TP računalnik. Če ta lahko opravlja kvazi istočasno samo eno nalogo, pomeni to, da kadar komunicira ne more opravljati drugih nalog in obratno. Kadar opravlja druge naloge je zaseden za dohodne pozive. Kadar je obremenitev računalnika z drugimi posli velika, je to za uporabnike zelo neprijetno, ker je zasedenost zelo pogosta. To neprijetnost pa lahko odpravimo z elektronskim poštnim nabiralnikom, ki ga namestimo v to namenskem centru. Ta center je računalnik, ki lahko sprejema hkrati več sporočil in jih hrani v svojem pomnilniku - elektronskih poštnih nabiralnikih uporabnikov. Ta sporočila prevzemajo nato uporabniki v svojem prostem času s klicanjem svojega elektronskega poštnega nabiralnika na iniciativo računalnika ali na lastno iniciativo.

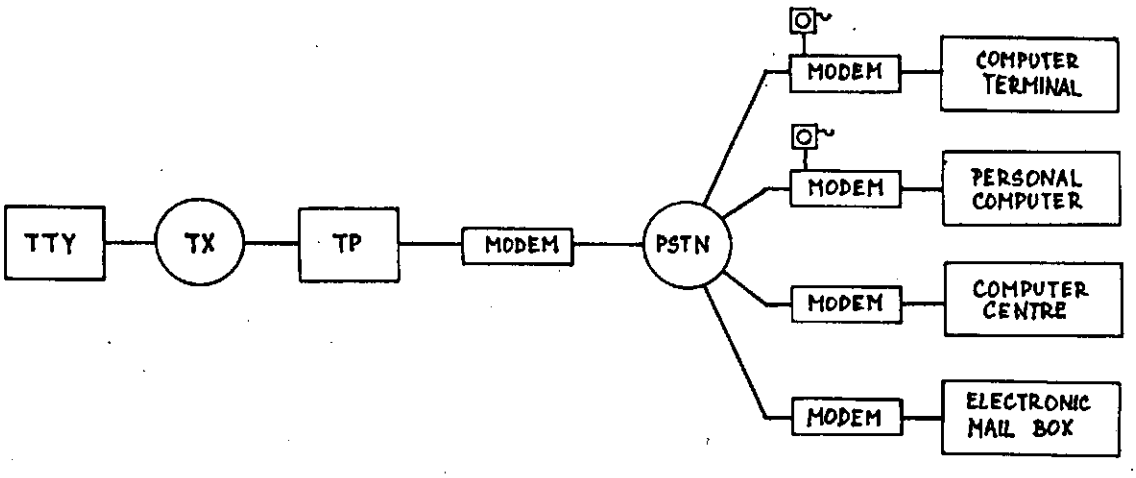
Pomanjkljivost te variante je, da je v tem primeru delovanje tranzitnega pretvornika zelo odvisno od programske opreme uporabnikovega računalnika. Ta predstavlja skupno s TX vmesnikom neločljivo povezano celoto, ki lahko le kot celota pridobi PTT atest.

Druga varianta, ki nima te pomanjkljivosti, je bolj kompliciran tranzitni pretvornik, ki ga prikazuje slika 6. Ta TP je možno s pozivom daljinsko vklopiti. TP se samostojno odziva na dohodne pozive, medtem, ko lahko nanj priključeni računalnik istočasno lokalno obratuje. TP tudi poskrbi za shranitev in ohranitev sprejetih sporočil dokler jih uporabnik ne pregleda. To se lahko zgodi na različne načine. Realno gledano sta možna naslednja dva načina:
- z nalaganjem dospelih sporočil v RAM tranzitnega pretvornika v katerem ga s pomočjo akumulatorskega napajanja ohranimo do prenosa v disk računalnika na iniciativo računalnika,
- s prenosom teh sporočil na printer, ki je povezan z računalnikom preko TP tako, da ga ta lahko vedno prevzame s pomočjo komutacijske enote KE v TP.

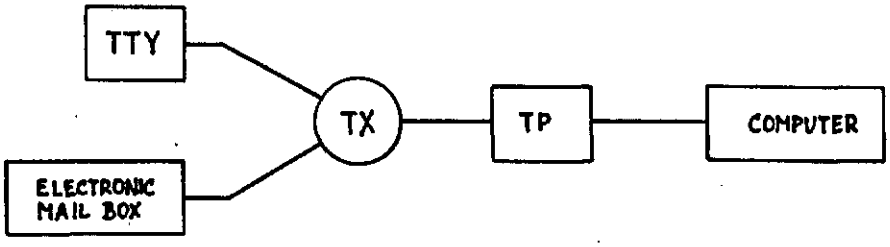
V obeh načinih se kaže tranzitni pretvornik kot zaseden, ko se napolni razpoložljivi RAM, ali pa če zmanjka papirja v printerju. Prvi način je ugodnejši od drugega, ker se v drugem primeru na papirju mešajo izpisi dohodnih sporočil z izpisi računalnika. Poleg tega je manipulacijska sposobnost podatkov na papirju manjša, kot tistih na disku.

ZAKLJUČEK

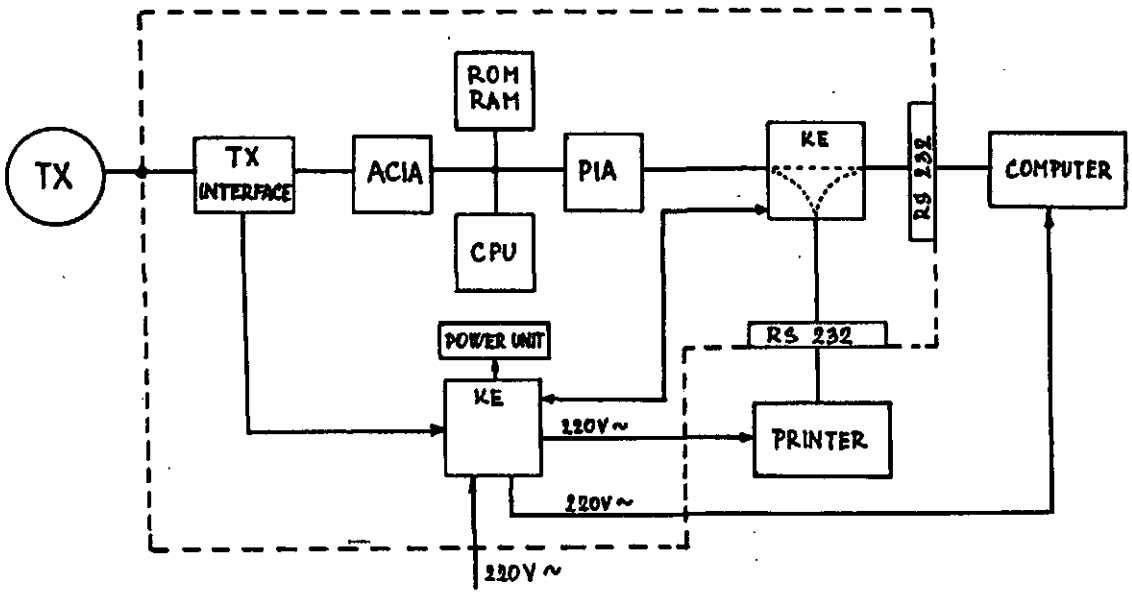
Z uvedbo tranzitnih pretvornikov, ki predstavljajo relativno majhno investicijo lahko takoj



Slika 4. Povezava računalnikov priključenih na PSTN s TX omrežjem.



Slika 5. Povezava računalnika direktno s TX omrežjem.



Slika 6. Tranzitni pretvornik za direktno priključitev računalnika na TX omrežje.

znatno povečamo informacijske pretoke na obstoječih telekomunikacijskih omrežjih. Zato bi morali stremeti, da bo ta oprema postala standardni sestavni del obstoječih telefonskih in telegrafskih central ter računalniških delovnih postaj. Na istem principu pa bi morali zagotoviti tudi prehod v paketno komutirano podatkovno omrežje in komunikacijo s teleteks terminali, že zaradi komunicitanja s temi napravami v tujini.

Literatura

[1] J.Massmann: Genau betrachtet RS 232/V.24 Schnittstelle, 64'er, 5 Mai 1985.

UVAJANJE TELEINFORMATSKIH STORITEV V PABX

MIROSLAV MARC
Iskra Telematika Kranj

UDK: 681.3.007

Prispevek opisuje razmere na področju govornih in negovornih uslug v naročniških telefonskih centralah. Poleg tega nakazuje trende nadaljnjega razvoja v smeri integracije različnih storitev (PABX, telefonskih garnitur, interfonskih naprav, lokalnih mrež) v okviru modernih teleinformatskih sistemov.

This article gives a description of conditions in the field of voice and non voice services in PABX. Besides, it points out future development trends in integration of different services (PABX, KEY, INTERPHONE SYSTEMS and LOCAL AREA NETWORKS) with modern teleinformation systems.

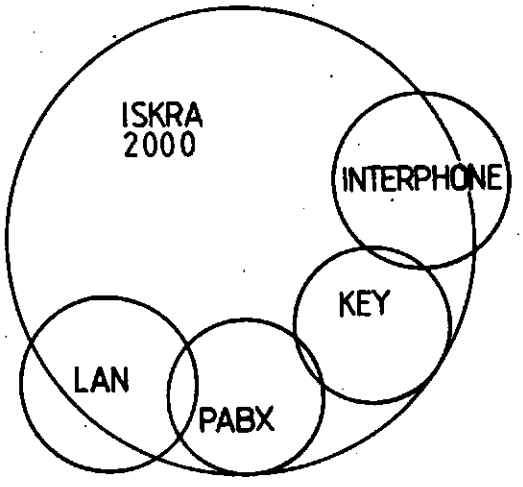
1. UVOD

Moderni poslovni sistemi (podjetja, banke, biroji, ustanove ...) že danes potrebujejo (v bodočnosti pa vedno bolj) zmogljive, praktične in hitre storitve kot so prenos GOVORA, TEKSTOV, PODATKOV in SLIKE.

Za reševanje omenjenih potreb se na tržišču pojavljajo naslednje grupacije naprav: zasebne telefonske centrale (PABX), telefonske sekretarske garniture (key systems), interfonske naprave (interphone) in lokalne računalniške mreže (LAN). Osnovne funkcije (storitve) posameznih naprav so v sistemu ISKRA 2000 integrirane na ta način, da bo uporabniku nudeno izvrševanje vseh zgoraj omenjenih storitev na enoten in kar je najpomembnejše na enostaven "uporabniško prijazen" način.

Gledano iz uporabniškega stališča tako postanejo funkcije (storitve) posameznih naprav: naročniški telefonskih central (PABX), telefonskih garnitur (KEY), interfonskih naprav (INTERPHONE) in lokalnih mrež (LAN) podmnožice storitev, ki jih rabijo moderni poslovni sistemi in jihrešujejo moderna vozlišča.

Vozlišče, na katero so priključeni razni terminali z modernimi storitvami ima tudi izhod (gataway) v javne in zasebne telefonske in netelefonske (tekst in data) mreže.



Slika 1: Integracija govornih in negovornih storitev

2. GOVORNE IN NEGOVORNE STORITVE NA PABX

2.1. Prognoza razvoja

V svetovni literaturi se pojavlja vse več ocen (prognoz) nadaljnjega razvoja na področju teleinformatskih storitev. Za primerjavo navajamo podatke ameriške consulting firme "Venture Development Corporation" (marec 1984).

	PRODAJA V MLD $				RAST
	1982		1987		87/82
STORITEV	VREDNOST($)	DELEŽ(%)	VREDNOST($)	DELEŽ(%)	
LAN+PABX	3180	100	5820	100	13
GOVOR/PABX	3110	98	5000	86	10
NEGOVOR/PABX	30	1	320	5	60
LAN	40	1	500	9	65

Tabela 1: Napoved razvoja teleinformatskih storitev

Iz tabele je razvidno, da so v letu 1982 v ZDA govorne storitve pomenile 98% vse prodaje, rast negovornih storitev pa je strma (preko 60% letno). Na drugi strani je rast negovornih storitev na naročniških telefonskih centralah (PABX) in lokalnih računalniških mrežah (LAN) izenačena.

Če primerjamo tudi drugo literaturo, analize in trende, ugotavljamo naslednje smeri razvoja svetovnega tržišča telekomunikacij in računalništva.

Na področju klasičnih telekomunikacij (tj. govornih storitev) bo po letu 1985 letna rast le 6,5%, saj je poleg že dosežene relativne nasičenosti tržišča instalirana oprema le "moralno" zastarela, sicer pa investicijsko draga. Na področju računalništva lahko računamo tudi le z nekaj nad 10% letno rastjo (sama programska oprema ima višjo rast). Ker pa je letna rast novega tržišča (negovornih storitev) izredno optimistična (preko 60%), prihaja do velikih vlaganj v razvoj novih sistemov.

Telekomunikacijski sistemi se vse bolj funkcionalno dopolnjujejo s pisarniško terminalsko opremo, ki mora protokolirati, distributirati in na pravem mestu arhivirati informacije.

Tak razvoj sili proizvajalce, ki so doslej nastopali samostojno na računalniškem oz. telekomunikacijskem tržišču v povezovanje (ATT - PHILIPS, OLIVETTI, IBM - ROLM, ERICSON - DATA SAAB itd...)

Na drugi strani vodi v enoten koncept proizvodnje naprav, sistemov za krmiljenje in urejanje informacijskih tokov tudi tehnološki razvoj sam. Ta razvoj pa je osnovan na integriranem digitalnem telekomunikacijskem omrežju (IDN), vključuje pa prvine razpršenega (distribuniranega) podatkovnega omrežja. Preoblikovanje raznih telekomunikacijskih sporočil vseh oblik v enotni (binarni) zapis pomeni digitalizacijo sporočil in s tem možnost univerzalnosti opreme. Ker se računalništvo na drugi strani razvija v smeri distribuiranih sistemov, narekuje to nujnost razvoja omrežij za medsebojno komuniciranje le-teh.

S proizvodnega stališča se torej oblikuje tehnološko enotno področje, ki združuje do nedavna ločeni programski področji: telekomunikacijski sistemi in poslovni računalniški sistemi.

Cilj razvoja je vspostaviti globalni sistem za komuniciranje med ljudmi, ki bo omogočal izmenjavo sporočil v najprimernejši obliki, ureditvi, vsebini in zgoščenosti; omogočal pa bo tudi diskretnost informacijskih tokov znotraj korespondentov.

2.2. Tri generacije elektronskih naročniških central (EPABX)

Glavna prednost in bistvena lastnost elektronskih naročniških central v primerjavi z elektromehanskimi je ta, da so računalniško krmiljene (SPC - Stored Program Control), kar jim omogoča bistveno boljše funkcijske in vzdrževalne možnosti. Lažje je tudi povezovanje v mreže (posebne ali javne).

Elektronske naročniške centrale lahko delimo v tri generacije zlasti z vidika obravnavanja negovornih storitev.

- Prva generacija

Pojavi se v sredini sedemdesetih let.
Značilno za prvo generacijo je, da so centrale namenjene predvsem in samo za govorne storitve. Značilna tehnologija prve generacije je analogna s prostorskim multipleksom (stikalno matriko). To je tudi zadovoljivo ustrezalo potrebam govornih storitev naj bo v lokalu ali vključitvi v javno mrežo.
Glavne lastnosti, ki jih nudi ta generacija, so izboljšane (razširjene) naročniške funkcije in centralizirana obdelava vzdrževalnih funkcij.

- Druga generacija

Pojavi se v prvi polovici osemdesetih let.
Naročniške centrale druge generacije so dizajnirane z namenom integracije govornih in negovornih storitev znotraj enega sistema. S tem odpada potreba po izgrajevanju lokalnih mrež za medsebojno komuniciranje podatkovnih terminalov. Tehnološka značilnost druge generacije je digitalizacija s časovnim multipleksom (PCM). Priključitev naročniškega terminala je lahko izvedena z analognim prenosom (telefon) ali z digitalnim, kjer lahko priključimo hiter podatkovni terminal ali celo računalnik. PCM kanal (64 K bit/sek) potegnemo do naročnika običajno štirižično.
Te centrale lahko povezujemo v široko paleto mrež za govor ali podatke tako z analognim kot digitalnim prenosom.

- Tretja generacija

Tretjo generacijo proizvajalci najavljajo v drugi polovici osemdesetih let.
Elektronske naročniške centrale tretje generacije prinašajo integracijo govora in podatkov znotraj enega kanala od multipleksa centrale vse do naročniškega terminala. Vsak terminalni priključek ima 144 K bitni kanal (2B + D) in tako lahko istočasno in neodvisno vzpostavlja govorne in negovorne komunikacije. Seveda imajo te centrale dodatne stikalne in naročniške funkcije za potrebe negovornih storitev.

Centrale tretje generacije nudijo možnost istočasnega vključevanja v govorne (tokokrogovne) mreže in podatkovne (paketne) mreže.
V tabeli 2 imamo zbrane tipične lastnosti posamezne generacije:

UPORABNIŠKE STORITVE \ GENERACIJA	PRVA	DRUGA	TRETJA
SAMO GOVOR	+	+	x
GOVOR ALI PODATKI	–	+	x
ISTOČASNO GOVOR IN PODATKI	–	+	+

\+ osnovni namen
x možno, vendar drago
* možno z okrnjenimi funkcijami

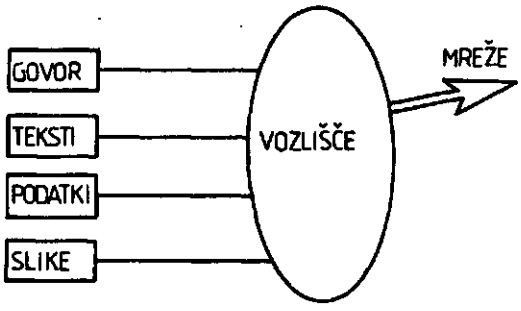
Tabela 2: Tipične lastnosti posamezne generacije

Seveda se posamezne generacije razvijajo dalje tako v pogledu širine funkcij kot površinske integracije. Zlasti za drugo generacijo je značilno postopno povečevanje števila podatkovnih terminalov (razširitve), kar je praktično z vidika cene in uporabnosti.

Centrale tretje generacije postajajo zanimive šele v zelo visoko tehnočno razvitem okolju. To je v pisarni "bodočnosti", ko ima velika večina uporabnikov (priključkov) *na delovni mizi poleg* telefona tudi podatkovni terminal, ki rabi istočasno z govorom visoko zmogljiv komunikacijski kanal do ostalih naročnikov.

3. MODERNE KOMUNIKACIJSKE STORITVE

Teleinformatske storitve, katerih integracijo lahko pričakujemo v bližnji prihodnosti:



Slika 2: Pričakovana integracija teleinformatskih storitev v bližnji bodočnosti

3.1. Govorne storitve

Končni cilj vseh govornih storitev je razgovor med dvema ali več udeleženci v zvezi.

Na strani telefonskega aparata ločimo standardne telefone (poziv, izbiranje in govor), posebne telefonske aparate (dodatne možnosti: regulacija jakosti, optične indikacije, spomin za izbiranje ipd...) in inteligentne govorne telefonske aparate (prikazovanje številk, tarife, ure, komunikacija pri položeni slušalki, avtomatski odzivnik, posredovanje telemetrijskih podatkov itd...)

Z vidika telefonske centrale, ki obravnava postopke za vspostavljanje zvez, poznamo naslednje skupine funkcij:
- možnosti telefonskega aparata
- možnosti posredovalnega mesta
- način iskanja

- omejitve in prestrezanje
- nočne službe
- promet in statistike i.t.d.

Osnovne lastnosti telefonskih (sekretarskih) garnitur so:
- priključitev več telefonskih aparatov na eno ali nekaj tele fonskih linij
- hitro medsebojno pozivanje telefonskih aparatov v garnituri
- pregled nad stanjem zunanjih linij (prosto, zasedeno, čakanje)
- predaja zunanje zveze iz enega na drug aparat v garnituri
- konferenčne zveze i.t.d.

3.2. Tekstovne storitve

Še pred električnim prenosom govora na daljavo se je pojavil prenos sporočil z MORSE-jevimi napravami. Nadaljevanje je sledilo z uvedbo teleprinterjev na start-stop principu, ki predstavlja pravi začetek tekstovnih uslug.

Danes se že vpeljuje nova storitev za prenos tekstov imenovana TELETEKS. Glavna prednost teleteksa je posodobitev, poenostavitev in pocenitev poslovanja v delovnih organizacijah in drugod. Gre za moderen sistem pripravljanja, pisanja, arhiviranja, pošiljanja in sprejemanja poslovne korespondence. Pričakuje se, da bo razvoj te storitve imel tudi velik vpliv na telex, faksimile in videoteks.

Osnovne značilnosti teleteks naprave so:
- oddajanje, sprejem in priprava dokumentov
- na sprejemni strani je dokument po svoji vsebini, razdelitvi teksta in po formatu popolnoma identičen dokumentu na oddajni strani
- priprava teksta se opravlja v lokalnem obratovanju
- informacija med dvema napravama se prenaša iz spomina oddajne v spomin sprejemne naprave
- hitrosti delovanja so različne: 2400, 4800, 9600 bit/sek

3.3. Podatkovne storitve

Spekter podatkovnih storitev je zelo širok. Začenja se pri standardnem telefonskem aparatu, ki mu dodajamo omejene možnosti posredovanja podatkov do centrale in nazaj. Imenujemo ga lahko podatkovni telefon, kar pomeni, da uporabniku poleg prenosa govora omogoča prenos tudi podatkov kot so na primer: čas, porabljena vsota, stanje na žiro računu, telemetrijske podatke ipd. Sem spadajo tudi razna merilna mesta, tipala in prikazovalniki.

Naslednji (čisti) podatkovni terminal je pasivni računalniški terminal. Sledijo mikroračunalniške delovne postaje in računalniški sistemi sami.

Kot vhodno izhodne podatkovne enote se najpogosteje uporabljajo video zasloni s tastaturami in različnimi pisalniki. Obstajajo tudi tako imenovani konzolni terminali, ki se uporabljajo ob večjih sistemih in prevzamejo tudi vlogo tiskalnika. Terminal imenujemo "inteligentni", ko ima možnost procesiranja podatkov in programov.

S stališča centrale kot vozliščnega centra pravzaprav ni bistvene razlike kakšen podatkovni terminal je priključen. Predvsem gre za način vzpostavljanja in rušenja zvez ter za prenosne hitrosti.

Ločimo asinhrone in sinhrone prenose podatkov. V asinhronem načinu prenosa si podatki sledijo v naključnih časovnih intervalih, v sinhronem pa si podatki sledijo eden za drugim brez vmesnega presledka za kontrolo prenosa.

Komunikacijski protokoli določajo nabor dogovorov med dvema enotama. Dogovori točno določajo sintakso, semantiko in časovno zaporedje izmenjave informacij.

V uporabi je več tipov protokolov (RS 232, BSC, SDLC itd.). Protokol HDLC (oziroma X.25) je standardiziran s strani CCITT in predstavlja osnovo za gradnjo ISDN omrežij.

3.4. Prenos slik

Pod to grupacijo razumemo storitve prenosa slik, risb, skic, skratka točk po koordinatnem principu, za razliko od tekstov kjer posamezni znaki (npr. črke) dobijo svojo šifro.

Prvi in najbolj razširjen predstavnik je FAKSIMILE, ki omogoča prenos dokumentov, slik ali skic na daljavo po koordinatnem principu (točke).

Zvezo z željenim naročnikom normalno vzpostavimo s telefonskim aparatom ali pa z vgrajenim setom za vzpostavljanje zvez. Zveza se ruši avtomatsko po koncu prenosa slike ali dokumenta.

Največ modernih faksimile naprav je izdelanih po priporočilih CCITT G2 in G3 s prenosnim časom 3 minute oz. cca 40 sekund za stran. Prenosni medij so javne ali najete telefonske linije.

Prenosne hitrosti so 2400, 4800, 7200 ali 9600 bit/sek. Obstajajo pa tudi digitalne faksimile, naprave s hitrostmi 16 in 48 kbit/sek.

Naslednji predstavnik pri prenosu slik so elektronske SKICIRKE (SKETCH PHONE). Uporabljajo se za pošiljanje skic in opomb na daljavo, medtem ko telefonska naročnika govorita po telefonu. Naprava vsebuje zaslon, ploščo z elektronskim peresom in včasih napravo za kopiranje.

Sistem deluje v govornem področju med 1.550 in 1.950 Hz po postopku semi duplex (CCITT V21). Torej ga lahko priključimo na normalno telefonsko linijo. Hitrost prenosa je 200 b/sek.

4. RAZVOJ SISTEMOV

4.1. Integracija govornih funkcij v modernih PABX

Za zadovoljevanje različnih potreb po govornem komuniciranju obstajajo na tržišču naročniške telefonske centrale (PABX), telefonske garniture (KEY) in interfonske naprave (INTERPHONE).

Izhajajoč iz ugotovitev, v poglavju 2, o smeri tehnološkega razvoja EPABX je normalno, da se različne uporabniške funkcije lahko realizirajo znotraj enega

sistema. V svetu se pojavljajo tudi novi nazivi, npr.: kombinirani sistemi ali hibridni sistemi.

Če pogledamo tipične funkcije oz. razlike posameznega sistema (PABX, KEY, INTERPHONE) vidimo, da so znotraj centralne krmilne enote (centrale) vse izvedljive. Z uporabniškega vidika so različni le telefonski terminali; za relizacijo PABX funkcij so običajni telefonski aparati, za KEY funkcije telefonski aparat garnitur (sekretarski) in za INTERPHONE-ske funkcije "hands free" telefonski aparati, za katere v slovenščini uporabljamo različne izraze: prostoročni, glasnogovoreči ali dupleksni telefonski aparati.

S tem so realizirane vse tipične funkcije posameznih sistemov (povezava, krmiljenje, izbiranje, signalizacije na aparatu, ozvočenje, omejitve, iskanje oseb, manipulacije med zvezo, brezročni odziv, dodatne informacije na aparatu itd...)

Nobenega razloga ni, da se funkcije vseh treh sistemov v prihodnosti ne bi združevale. S tem bo uporabnik z nabavo enega sistema lahko zadovoljivo reševal svoje specifične potrebe na enostaven in uporabniško prijazen način. To je tudi prvi korak v smeri integracije različnih storitev znotraj enega sistema, le-ta pa je že v teku.

4.2. Primerjava med PABX in LAN

Avtomatizacija pisarn bo glavno gibalo nadaljnega razvoja komunikacijskih naprav. Večino zahtevanih parametrov komunikacijskega prometa vsebujejo PABX. Hkrati pa se razvijajo tudi lokalne računalniške mreže (LAN), ki pa imajo drug spekter aplikacij.
V pisarniškem okolju imamo torej PABX za izvrševanje govornega in podatkovnega komuniciranja ter LAN za izredno hiter prenos podatkov.

Obe rešitvi medsebojno povezujeta naročnike, PABX s prekljapljanjem linij, LAN pa z usmerjanjem paketov. V PABX je procedura za vzpostavljanje zveze časovno ločena od samega prenosa podatkov (v centralah tretje generacije to ne velja več), v lokalni računalniški mreži pa je vzpostavljanje zveze in prenos podatkov časovno združeno. Zato imata mreži tudi različne protokole za vzpostavljanje zvez.

LAN imajo občutno prednost le pri hitrosti prenosa podatkov. PABX omogočajo prenos podatkov do hitrosti 64 Kbit/sek (le v posebnih primerih do 2 Mbit/sek), medtem ko LAN omogočajo hitrosti do 10 Mbit/sek.

Ker bo govor še nadalje glavna komponenta komunikacij, bodo ta segment dobro pokrivale PABX, ki bodo omogočale tudi prenos podatkovnih sporočil, katerih hitrost je običajno izpod 64 Kbit/sek.

Nadaljna smer razvoja PABX je vsekakor v distribuirani arhitekturi, popolni digitalizaciji, razvoju inteligentnih terminalov kot sestavnih delov PABX, večjih zahtevah po zanesljivosti in popolni tehnični integraciji vseh oblik sporočil. Nadalje se kaže kot zelo uspešna povezava obeh principov tako, da se mrežni arhitekturi (LAN in PABX) povezujeta in dopolnjujeta v delovanju.

5. ZAKLJUČEK

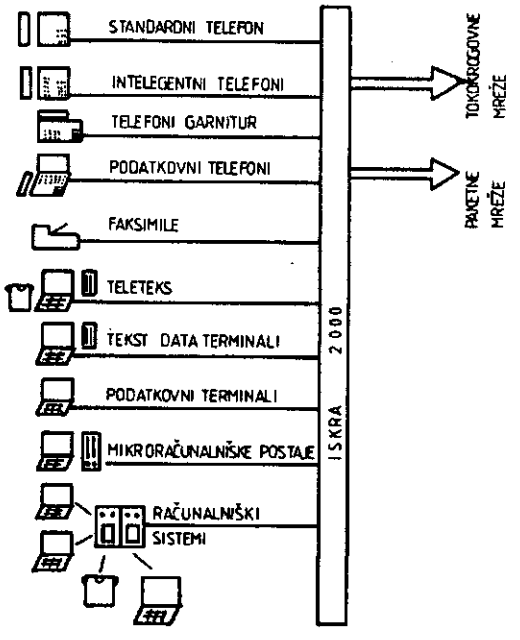
Informacijska družba, v katero prihajajo razvite družbe, zahteva vse večje obdelave in prenose informacij. V preteklosti so se informacije obdelovale in prenašale ločeno. Računalniki so informacije obdelovali. Za prenos informacij pa so se uporabljali telefonski, telegrafski in podobni sistemi. Razvoj računalnikov je zahteval medsebojno povezovanje računalnikov v računalniške mreže. Ravno tako pa je prenos informacij po telefonsko telegrafskih sistemih začel uvajati nove funkcije, ki zahtevajo vse več obdelav. Vse večje prekrivanje računalniških in prenosno komutacijskih funkcij je privedlo do združevanja obeh v teleinformatske sisteme, ki imajo funkcijo tako obdelave kot komutacije in prenosa informacij.

Posamezne storitve (teleteks,videoteks), ki so bile v dosedanjih sistemih zaradi pomankljivosti bodisi računalniških bodisi komutacijskih sistemov zelo težko izvedljive, dobivajo svoje mesto v sodobnih teleinformatskih sistemih.

Naročniške centrale, telefonske garniture in interfonske naprave rešujejo vsaka del specifičnih zahtev. S tehnološkim razvojem modernih računalniško krmiljenih central postajajo specifičnosti vseh treh sistemov enostavno rešljive znotraj enega sistema. Najpomembnejša prednost je uporabniško prijazna možnost uporabe.

V vsakdanjem poslovnem življenju se pojavlja vse več informacij (podatkov). Te informacije bistveno olajšajo in racionalizirajo odločitve v kolikor so dostopne v ustrezni obliki ob pravem času na pravem mestu. Dosedanje pisane informacije so se pojavljale predvsem v obliki dopisov. Sodobna komunikacijska sredstva lahko prevzamejo večino funkcij klasičnega dopisovanja, obenem pa omogočajo najrazličnejše obdelave informacij tako, da nam izbirajo samo tiste informacije, ki nas v danem trenutku zanimajo, medsebojno primerjajo različne podatke, izvajajo izračune, oblikujejo informacije v najustreznejšo obliko (diagrami), prenašajo informacijo na ustrezna mesta itd. Bistveno se poveča hitrost obdelave in prenosa informacij.

Za zaključek podajamo sliko možnega priključevanja terminalov v teleinformatski sistem kot jih predvideva sistem ISKRA 2000.



UVAJANJE NOVIH FUNKCIJ V TELEINFORMATSKE SISTEME

Iztok Tvrdy(1), Franc Rozman(2), Rajko Sabo(1), Helena Tvrdy(1)

(1) Institut Jožef Stefan, Jamova 39, Ljubljana
(2) ISKRA Telematika, Ljubljanska 24, Kranj

UDK: 681.3.007

Teleinformatske funkcije so sestavni del porajajočih se teleinformatskih sistemov. Funkcije lahko zaenkrat le načrtujemo, niso pa še realizirane. Podajamo pregled dosedanjega razvoja telefonskih in računalniških sistemov, predviden razvoj teleinformatskih sistemov in z njimi povezan razvoj funkcij.

INTRODUCING NEW FUNCTIONS IN TELEINFORMATIC SYSTEMS. Teleinformatic functions are a part of the nascent teleinformatic systems. These functions are not realized yet. Overview of the telephone and computer systems development existing till now, future development of teleinformatic systems and proper functions are given in this paper.

Uvod

Telefonska omrežja se po svetu razvijajo v integrirana digitalna omrežja (Integrated Digital Network - IDN) s 64 kbit/s komutacijskimi kanali. Sedaj veljavni cilj razvoja materialne opreme pa predstavlja univerzalno digitalno komunikacijsko omrežje (Integrated Service Digital Network - ISDN). Veliki napori in sredstva se vlagajo v določevanje najustreznejše poti za optimalno integracijo telefonskih in ostalih uslug. Izvedba kompleksnih teleinformatskih sistemov nad univerzalnim digitalnim komunikacijskim omrežjem pa je cilj razvoja teh uslug. Razvoj bo tekel postopoma, tako da bo pri razvoju IDN kot tudi pri razvoju računalniških mrež potrebno poskrbeti za fleksibilno in postopno uvajanje novih funkcij v smislu združenega teleinformatskega sistema.

Da bi razvoj teleinformatskih funkcij lahko čim točneje napovedali, moramo pregledati dosedanji razvoj telefonskih in računalniških sistemov.

Razvoj telefonskih sistemov

Tehnologija analognih telefonskih signalizacij omogoča skromen nabor telefonskih funkcij. Omogoča predvsem posredovane in avtomatske govorne zveze, vzpostavljene na osnovi impulznega ali MFC (tonskega) izbiranja.

Da bi se v analognih telefonskih sistemih povečale funkcijske možnosti, so se vzporedno s telefonskimi sistemi pojavile sekretarske naprave, ki omogočajo predvsem hitrejše vzpostavljanje zvez na krajše razdalje, ter interfonski sistemi, katerih prednost je brezročno odzivanje na pozive.

Digitalna tehnologija z digitalnim telefonskim aparatom združuje vse te funkcije v enem sistemu. Funkcije se izvajajo na mnogo večjih geografskih razdaljah. Zaradi bogatejše informacijske povezanosti omogoča uvajanje dodatnih funkcij, ki v analogni tehnologiji niso poznane.

Razvoj računalniških sistemov

Računalniki so se v začetku pojavili kot samostojne med seboj ločene enote s paketnim (batch) načinom delovanja. Sledili so računalniki s terminali, ki so omogočali interaktivni pristop. Z večanjem števila računalniških aplikacij se je večala potreba po računalniških mrežah. Naslednja faza razvoja računalnikov pa je razporeditev podatkovnih terminalov na poljubno delovno mesto.

Prve računalniške aplikacije so bile le obdelave podatkov. Z gradnjo računalniških mrež, to je z razporeditvijo in povezavo računalniških terminalov in obdelav po širšem območju, pa prihaja vse bolj do izraza potreba po prenosu podatkov med terminali. Poleg samih "pravih" numeričnih podatkov se vse bolj prenašajo tudi teksti in slike.

Teleinformatski sistemi

S hkratnim razvojem telefonije in računalništva je prišlo do potrebe po združeni telefonski in računalniški povezavi med istimi uporabniki. Da ne bi vzporedno gradili dveh ločenih komunikacijskih omrežij, se razvija teleinformatska mreža v digitalni tehnologiji, ki naj bi prevzela funkcijo telefonskega in računalniškega povezovanja uporabnikov. Teleinformatski sistem je torej sistem za komutiranje, prenos in obdelavo govornih, podatkovnih, tekstovnih in slikovnih informacij.

Teleinformatski sistemi bodo v splošnem nudili usluge tako v javnem kot v privatnem omrežju. V francoskem govornem področju pa te sisteme običajno ločijo na teleinformatske sisteme, ki nudijo javni servis, in na telematske sisteme, ki nudijo servis v privatnih omrežjih.

Razvoj teleinformatskih funkcij

Na splošno smer razvoja teleinformatskih funkcij vplivajo naslednji faktorji:

- preverjena uporabnost klasičnih telefonskih funkcij
  Vse telefonske, interfonske in sekretarske funkcije, ki so se izkazale kot uporabniško zanimive, se bodo prenesle v teleinformatske sisteme (na primer v digitalni telefon).

- razširjene tehnološke možnosti digitalnih sistemov
  Funkcije, ki so se uporabljale le v okviru ene centrale oziroma ene sekretarske naprave, se bodo izvajale v okviru celotne teleinformatske mreže oziroma na posameznih delih teleinformatske mreže.

- preverjena uporabnost računalniških aplikacij
  Računalniške aplikacije, ki so se izkazale kot uporabne za širši krog uporabnikov, se bodo izvajale s pomočjo poljubnega teleinformatskega terminala. Le-ta bo seveda moral imeti zato določene standardne lastnosti.

- varovanje in zaščita podatkov
  Povečan obseg podatkovnih baz in poenostavljen ter hitrejši dostop do podatkov bo zahteval veliko dela v zvezi z zbiranjem in vzdrževanjem podatkov, prav tako pa tudi skrbnost pri posredovanju podatkov uporabnikom. Onemogočiti bo treba možnost zlorabe ali uničenja podatkov, zato bo imel teleinformatski sistem varovalni sistem ugotavljanja istovetnosti uporabnikov podatkov, uporabe gesel, razvrščanja uporabnikov po razredih glede dostopa do podatkov in drugo.

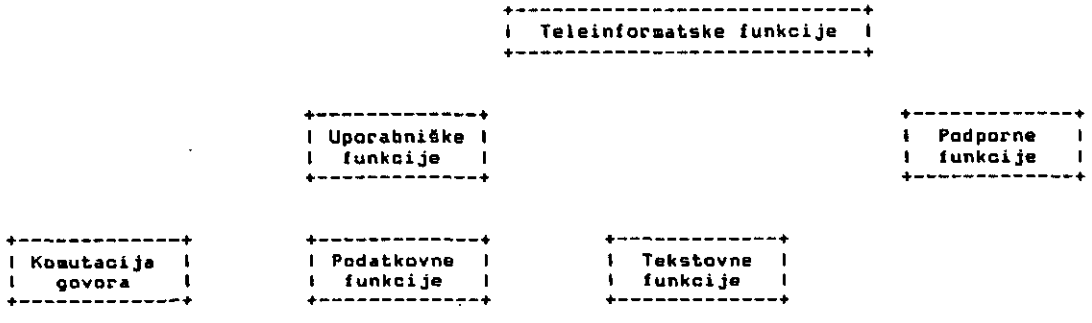
- širok dostop do podatkovnih terminalov
  Povzročil bo hitro naraščanje podatkovnih baz, s tem pa tudi funkcij, vezanih na podatkovne baze (videoteks, urejanje teksta, iskanje podatkov, ...). Dostopnost uporabnikov do terminalov bo bistveno povečala pretok informacij med končnimi terminali (teleteks, faksimile, elektronska pošta, videokonferenca, ...).

- procesorsko krmiljeni terminali
  Procesorsko krmiljeni terminali bodo povzročili porazdelitev funkcij po posameznih terminalih. V dosedanjih telefonskih sistemih smo vse telefonske funkcije v celoti izvajali v telefonski centrali, teleinformatski sistemi pa bodo vse več funkcij prenesli na enega ali več med seboj povezanih terminalov. Teleinformatski centrali bo zato ostalo le komutiranje različnih zvez.

Opredelitev teleinformatskih funkcij

Predhodno opisane smeri razvoja kažejo, da bomo že v bližnji prihodnosti morali realizirati veliko število funkcij. V dosedanjih telefonskih sistemih smo realizirali vse funkcije, ki jih je omogočala tehnologija. V teleinformatskih sistemih pa bomo morali predhodno ugotoviti uporabniško vrednost vsake posamezne funkcije v konkretnem okolju ter realizirati le tiste, ki so tam zanimive. Zahteve različnih okolij (bančništvo, proizvodnja, turizem, zdravstvo, ...) so med seboj različne. Teleinformatski sistemi pa bodo zanimivi le, če bodo zadovoljevali natanko tiste funkcije, ki jih določeno okolje potrebuje.

Opredeljevanja teleinformatskih funkcij se lotimo tako, da analiziramo uporabnost posameznih poznanih skupin funkcij, ki so razvidne s slike 1.



Slika 1. Dekompozicija teleinformatskih funkcij

Komutacija govora

Funkcije komutacije govora razdelimo na telefonske, sekretarske in interfonske funkcije [1,3,13].

Podatkovne funkcije

V grobem razdelimo podatkovne funkcije na:
- zajemanje in vzdrževanje podatkov,
- predstavitev podatkov,
- komutiranje in prenos podatkov ter
- obdelavo podatkov.

Zajemanje in vzdrževanje podatkov izvaja pooblaščeno strokovno osebje na predpisan način in z natančno določenimi dosegi preko računalniških vhodno-izhodnih enot, kot so terminali (CRT terminali, registrirne blagajne, čitalci magnetnih kartic), merilne naprave (števci električne energije, merilniki pretoka tekočin, termometri) in nadzorne naprave (računalniki za nadzor tehnoloških procesov).

Predstavitev podatkov izvajajo:
- terminali: CRT terminali, prikazovalniki (displayi), registratorji, instrumenti,
- krmilniki: krmilniki stikal, procesni računalniki, ...

Komutiranje in prenos podatkov
Pri prenosu podatkov lahko uporabimo večino funkcij, poznanih iz prenosa govornega signala (lokalna zveza, javna zveza, kratko izbiranje, prioritetna linija, preusmeritev poziva, ...). Ob tem pa moramo upoštevati dodatne zahteve po ekstremno dolgih oziroma po ekstremno kratkih zvezah, zanesljivosti pri prenosu podatkov, hitrosti prenosa, itd. Tem dodatnim zahtevam pri komutiranju podatkovnih zvez zadoščajo različne tehnološke rešitve prenosa in komutacije podatkovnih zvez (lokalne mreže, omrežja za prenos podatkov; preklapljanje linij (vodov), preklapljanje sporočil ali preklapljanje paketov).

Obdelava podatkov
Vsak proces (tehnološki, ekonomski, sociološki, ...) zahteva nabor zanj specifičnih obdelav. Takih aplikacij obstoja v svetu že sedaj ogromno, njihovo število in velikost pa stalno še naraščata. Na tem mestu ni smiselno naštevati posameznih aplikacij obdelav podatkov, zato podajmo le njihovo razdelitev po skupinah:
- krmiljenje tehnoloških procesov,
- bančniško poslovanje,
- avtomatizacija uradov,
- prometno - turistični informacijski sistemi,
- razvojni in podporni sistemi, in tako dalje.

Tekstovne funkcije (prenos tekstov, dokumentov in slik)

Funkcije prenosa tekstov, dokumentov in slik delimo na naslednje skupine:
- zajemanje in predstavitev tekstov,
- zajemanje in predstavitev dokumentov,
- zajemanje in predstavitev slik,
- hranjenje slik, tekstov in dokumentov ter
- prenos in komutiranje tekstov, slik in dokumentov.

Zajemanje in predstavitev tekstov izvajamo na teleteks terminalih, ki so lahko bodisi funkcijsko zelo okrnjeni, ali pa so kar kompleksni računalniški sistemi z bogatimi funkcijami. S teleteks terminali izvajamo:
- različne načine zajemanja tekstov,
- različne interpretacije črkovnih znakov,
- različne možnosti shranjevanja tekstov,
- oblikovanje tekstov, ...

Zajemanje in predstavitev dokumentov izvajamo s pomočjo faksimile naprav. Funkcije teh naprav so reprodukcija različnih formatov dokumentov, barvna ali črno-bela predstavitev prenešenih dokumentov, ...

Zajemanje slik omogočajo kamere, predstavitev slik pa različni monitorji oziroma kar teleinformatski terminali. Funkcije zajemanja slik so podobne funkcijam klasične foto-kino tehnike.

Funkcijo hranjenja tekstov, dokumentov in slik opravljajo videocentri oziroma drugi sistemi za upravljanje s podatkovnimi bazami. Način organiziranja podatkovnih baz je odvisen od obsega podatkov, zahtev za vnos novih podatkov v podatkovne baze, od zahtev za dostopnost podatkov, ...

Funkcija prenosa in komutiranja tekstov, dokumentov in slik je podobna funkciji prenosa govora in podatkov. Dodatne zahteve pa so pogojene s številom vzpostavljenih zvez, trajanjem vzpostavljenih zvez, prometnimi karakteristikami prenosnih poti, ... (videokonferenca).

Podporne funkcije

S podpornimi funkcijami teleinformatskega sistema, kot so polnjenje, administriranje, diagnostika, meritve, zaščita in tarifiranje urejamo zanesljivost in prilagodljivost sistema na prometne in funkcijske zahteve sistema ter obračunavanje uslug.

Trenutno stanje razvoja v svetu in pri nas

Po svetu so se nekatere od omenjenih funkcij že razvile, toda ne v okviru enotnega digitalnega omrežja. Prenos podatkov, videoteks in teleteks so funkcije, ki skoraj v vseh evropskih državah že že delujejo ali pa so vsaj v pripravah. Pri nas so te tri funkcije v pripravi ali v eksperimentalnem delovanju, seveda pa so zaenkrat med seboj še ločene. Tudi proizvajalcev ustreznih naprav je precej. Seveda teh naprav zaradi neuniverzalnosti ne moremo poimenovati za teleinformatske terminale.

Zadnje novice [9] potrjujejo, da so se na ameriškem tržišču poleg privatnih omrežij digitalnih naročniških telefonskih central (ROLM - CBX) že pojavili digitalni telefoni z vmesniki proti IBM PC kompatibilnim mikroračunalnikom in celo mikroračunalnik, v katerega je digitalni telefon že vgrajen (Juniper, Cedar). Te nove enote lahko smatramo kot poskus uvajanja določenega tipa teleinformatskih terminalov.

Predhodniki teh samostojnih delovnih postaj so terminali tipa ES.1 in ES.3 proizvajalca Zaisan, vendar pa le-ti temeljijo še na analogni podatkovni komunikaciji.

V Kanadi obstojijo podobni realizirani poskusi: Northern Telecom je dal na tržišče družino Displayphone, katere najnovejši predstavnik Displayphone Plus ima vgrajeno emulacijo protokola VT100 ter interni 1200 baudni modem.

Zanimiv je francoski eksperiment VELIZY: telefonski naročnik dobi poleg telefonskega aparata poseben terminal, s katerim dosega podatke v telefonskem imeniku, opravlja bančne transakcije in se hkrati vklaplja tudi v francoski videoteks TELETEL. Ta eksperiment že kaže na integracijo določenih uslug.

Iz druge smeri, iz čistega računalniškega okolja potekajo v okviru ESPRITa (skupni program EGS - Evropski Strateški Program za Raziskave v Informacijski Tehnologiji [4]) raziskave z operacijskim sistemom UNIX UNITED, to je distribuiranim sistemom UNIX v okolju medsebojnih povezav odprtih sistemov (OSI/ISO). Na eni strani nameravajo zgraditi povezavo med računalniki na podlagi referenčnega modela ISO OSI, na drugi strani pa uporabiti protokol X.25 za komunikacijo z javnim omrežjem. Pri tem naj bi bila programska oprema za komunikacijo čim bolj prenosljiva, to je neodvisna od računalnika in njegovega operacijskega sistema. Cilj raziskav je izdelava sistema, ki bi uporabniku bistveno olajšal delo na distribuiranih sistemih (dostop do poljubnega dela sistema, iskanje informacije, elektronska pošta, distribuirano editiranje tekstov, izmenjava programov, ...).

Na področju standardizacije komunikacij in prenosa podatkov se na mednarodnem področju (ISO, ECMA, CCITT) odvija živahna dejavnost - pred nedavnim so prejeli nove standarde MHF in X.400.

Zaključek

Referat podaja le grobo strukturo teleinformatskih funkcij z označitvijo njihovih osnovnih karakteristik. Vsak izdelek (telefonska centrala, terminal, računalnik, ...) pa bo imel natančno opredeljene vse funkcije, ki jih bo lahko izvajal v konkretnem teleinformatskem sistemu.

Teleinformatske funkcije ne bodo izvedene le na enem mestu, kot v dosedanjih telefonskih centralah, ampak porazdeljeno po komutacijskih sistemih in terminalih. Funkcije zajemanja tekstov bodo, na primer, realizirane v teleteks terminalu. Zato so te funkcije zanimive le s stališča načrtovanja teleteks terminala, popolnoma nezanimive pa so pri načrtovanju same teleinformatske centrale.

Z uvajanjem teleinformatskih sistemov meja med računalništvom in telekomunikacijami izginja; uporabnik pa se iz nedejavnega sprejemalca informacije, kakršno mu nudi sedanja obča tehnologija (pisana besedila, časopisi, radio, televizija), spreminja v tvornega udeleženca procesa dvostranske izmenjave informacij. Pri tem pa si ne želi Orwellovega Telekrana.

Literatura:

[1] Glossary of Terminology, PBX Systems, The Marketing Programs and Services Group inc., Gaithersburg, Maryland, 1981

[2] Savetovanje o uticaju novih službi na promenu tokova informacija, Zajednica jugoslovenskih PTT, Dubrovnik, 1984

[3] Tovarniški predpis TP-A.SO.100 Pregled funkcij naročniških telefonskih central, Iskra Telematika, DSSS - PRS, Standardizacija in dokumentacija, Kranj, 1984

[4] Bendayan J.: The "Unix United" Aspects of the I.E.S., A Distributed Unix Sistem in an "Open System Interconnection" Enviroment, Information Exchange System, Pilot Project Nr. 130, First ESPRIT Technical Week, Brussels, September 1984

[5] Cornell R. G., Stelte D. J.: Progress Towards Digital Subscriber Line Services and Signaling, IEEE Transactions on Communications, Vol. Com. 29, No. 11, November 1981

[6] Dorros I.: Telephone Nets Go Digital, IEEE Spectrum, April 1983

[7] Karavators P. W.: The Next Generation in Business Communications, Telecommunications, August 1983

[8] Keller D. A., Young F. P.: DIMENSION AIS/System 85 - The Next Generation Meeting Business Communications Needs, IEEE, 1983

[9] Moran T.: ROLM integrates IBM PC compatibility into its proprietary PBX network, Mini-Micro Systems, November 1984

[10] Rudov M. H.: Marketing ISDNs: Reach Out and Touch Someone's Pocketbook, Data Communications, McGraw-Hill, June 1984

[11] Takahashi T., Nagase H., Ueno T.: Function Enhancement of Digital Local Exchange to Accomodate Digital Terminals, IEEE, 1983

[12] Thomas J.: Electronic Business Telephone, GTE, Automatic Electronic Journal, March 1980

[13] Tvrdy I., Unk M., Canjko C., Dežman M., Tvrdy H.: Funkcije naročniških telefonskih central, Institut Jožef Stefan, IJS delovno poročilo DP-3333, Ljubljana, 1983

INTERAKTIVNI RAD POSREDSTVOM TELEX MREŽE

JELOVICA BUDIMIR
JADROAGENT - RIJEKA

UDK: 681.3.007

SAŽETAK: - U referatu je prikazan način uključivanja u informacioni sistem putem javne telex mreže. Ukazano je na mogućnost pristupa centralnoj bazi podataka u smislu njenog ažuriranja ali i postavljanja upita. Naglašene su specifičnosti u rješavanju načina pristupa, proizašle zbog sporosti u radu sa klasičnim teleprinterom. Nabrojene su i ostale mogućnosti rada kao što su prijem telexa ali i slanje običnih i cirkularnih obavijesti uz automatsko biranje brojeva.

ABSTRACT: - The telex combines all the requirements needed to make possible a dialogue between the user and a computer, namely:

- a keyboard enabling questions to be formulated or data called up.
- a print-out facility enabling replies to questions put to be received instantly.

The possibilities offered by the telex are therefore the same as can be with any real time computer terminal and can be categorized under three main headins:

- data acquisition and verification
- interrogation of central data bases
- automatic message transmission

UVOD

Za one informacione sisteme, koji ne gladuju za velikom količinom podataka (barem ne u nekim svojim dijelovima), ali zahtjevaju izuzetno velik broj punktova priključenih u kraćim intervalima vremena, dobro rješenje za prikupljanje informacija biti će korištenje međunarodne telex mreže. Ovako rješenje dati će izvanredne rezultate za informacione sisteme, npr. u agencijskim kućama, bez obzira da li su one turističke, putničke ili brodarske. Sve se one bave rezervacijom i prodajom prostora ili samo izdavanjem informacija, bez obzira da li u dvorani, igralištu, autobusu, brodu ili hotelu. Za sve njih vrijedi da im klijenti nisu uvijek unaprijed poznati. A svaki od njih sigurno posjeduje običan, stari i spori teleprinter.

U svijetu postoje rješenja koja pokrivaju ovo područje već više od jednog decenija. Možda začuđuje ovakav spoj dvije tehnologije iz praktički dva različita stoljeća, ali se on pokazao korisnim, naročito u agencijskom poslovanju, gdje je tehnička opremljenost različitih agenata upravo na tom-prapotopnom stupnju razvoja informatike.

U svrhu spajanja računala na telex mrežu, razvijen je hardware te software apliciran na računala DELTA 4850 (VAX 11). Programski paket dobio je zaštitni naziv

PROTOKOL EMULATOR ZA TELEX KOMUNIKACIJU ili skraćeno - P E T K O,

a sadržava tri osnovne funkcije:

- funkcija interakcije sa bazom podataka informacionog sistema. Ova funkcija pruža mogućnost interaktivnog rada u dva pristupa - upit na bazu i ažuriranje baze
- funkcija prijema teksta ili formatiranih izvještaja
- funkcija distribuiranja otvorenih tekstova i/ili izvještaja dobivenih iz različitih programa. U ovu funkciju dodano je auto-call svojstvo (sposobnost), pa je time čovjek osloboden od čestog i mukotrpnog pozivanja.

U postojeći informacioni sistem Jadroagenta nazvan "JACOB" (Jadroagent Computer Operating and Booking) će se projekt "PETKO" uklopiti za potrebe praćenja kretanja kontejnera.

OPIS SKLOPOVSKOG DIJELA

Sklopovski dio čine:

- procesor DELTA 4850
- asinhroni interface (RS 232)
- adapter za telex mrežu

Konstrukcija sklopa je takva da zadovoljava tehničke zahtjeve za vezanje na telex PTT mrežu s jedne strane, i zahtjeve za vezanje na RS 232 EIA CCITT standarde. Sklop ne vrši nikakvu logičku konverziju kodova, već samo naponsku i strujnu konverziju uz respektiranje polariteta strujnih i naponskih signala i vremenskog slijeda promjene tih signala.

P E T K O - kratki funkcionalni opis

PETKO je samostojeći (rezidentni) program koji omogućava interaktivni i batch režim rada informacionog sistema sa telex pretplatnicima. PETKO omogućava slanje niza tekstova (fileova) koji se kolektiraju u internoj listi čekanja, a redoslijed plasiranja na liniju regulira se prioritetima. Biranje brojeva je programski rješeno slanjem baudot koda prema TLX centrali. Pozivanje se vrši određeni broj puta, a zatim se trenutni poziv stornira i prebacuje na kraj liste čekanja. Kad nema tekstova u listi, program ima status prijema.

Izvještaje o primljenim i predanim teleksima PETKO šalje na konzolni pisač. Program se instalira u memoriju, te ostaje rezidentan kad nema ni prijema ni predaje. Njemu se ne dodjeljuje niti jedan terminal, već se po potrebi vrši "prijavljivanje" sa bilo kojeg terminala u svrhu ubacivanja novog teksta u listu čekanja ili izdavanje neke druge komande. Prijavljivanje je posredno, preko tzv. dispečer-programa koji vrši logičko spajanje između terminala i PETKA. Po završetku rada sa terminala, operater daje eksplicitnu naredbu za "odspajanje" terminala. Komunikacija sa PETKOM ostvaruje se preko niza komandi, kojima se iniciraju slijedeće funkcije:

1. Ubacivanje teksta u listu čekanja uz slijedeće param:
   - pozivni broj
   - naziv teksta (file name)
   - prioritet (0-7)
   - broj pokušaja nazivanja u bloku (1-10)
2. Izbacivanje teksta iz liste čekanja
3. Pregled liste čekanja
4. Trenutni prekid tekuće komunikacije
5. Stopiranje trenutnog transfera i prelaz na interaktivni rad
6. Povrat iz interaktivnog u batch rad
7. Stopiranje komunikacije nakon završetka tekućeg transf.
8. Prebacivanje u stanje prijema za određeni interval

Osim slanja PETKO ima i mogućnost primanja telexa, te njihovo logiranje u određena polja na disku. Svaki poziv pa do prekida linije tretira se kao jedna cjelina i kreira po jedno polje.

NAČIN PRISTUPA INFORMACIJAMA

Pristup informacijama preko telexa treba biti pažljivo obrađen:

a) zbog male brzine ispisa

b) zbog poludupleksnog načina rada

Ispisi moraju biti što kraći, a sumarni podaci uvijek staknuti prije nego što se izbaci lista pojedinačnih događaja ili stanja. Sumarno stanje obično sadržava podatak koliko će lista biti dugačka. Iz navedenih razloga se za svaki objekt promatranja odn.entitet moraju definirati različiti vidovi promatranja. Zahtjev je da vidovi budu što uži ali u mjeri u kojoj ostaje sačuvana njihova korisnost.
Za svaki vid se, nadalje, definiraju pojmovi (ključevi po kojima se daje sumarni pregled. Sumari su složeni tabelarno, a koordinate tabela obično predstavljaju vrijednosti novih ključeva po kojima se ide u daljnje razvrstavanje podataka.
Istovremeno, svi oni moraju zadovoljiti logički pogled korisnika informacionog sistema na podatke, pa se prema tome moraju definirati u suradnji s korisnicima.

Sekcija odgovora sa određenom grupom podataka počinje navođenjem svih ključnih pojmova, obzirom da se ključevi gornjih nivoa ne moraju ponavljati. Zatim se ispiše tabela sa sumarnim podacima za traženje skup ključeva. Ujedno se u tabeli nalaze novi ključni pojmovi (jedan ili više) sa pripadnim atributima. Na dnu sekcije se ponove novi ključni pojmovi, te se može nastaviti sa upitom.
Za onog tko prvi puta postavlja upite je ovaj sistem višeslojnog prikazivanja informacija zgodan, jer ga vodi do traženog cilja.

PRIMJER INTERAKTIVNOG RADA PRETPLATNIKA

Počinje nakon što pretplatnik dobije vezu te ukuca početnu naredbu i svoju šifru,

/ SET INTERACTIVE SIFRA

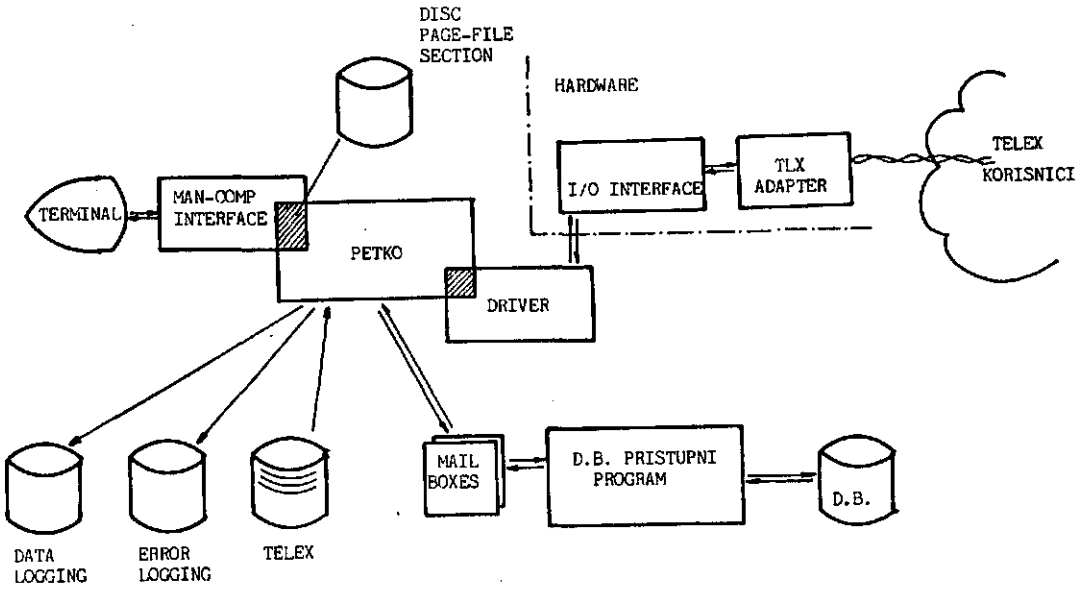
koja završava malteškim križem ( ).

Nakon izvršene provjere šifre, koja u kombinaciji sa odzivnikom mora odgovarati, sistem odgovara za:

ID (ENTITET) PRIMLJEN

(četiri podcrte označavaju prihvaćanje komande).

U slijedećem koraku se definira vrsta posla te entitet Vrsta posla može biti upit/transakcija, a entitet mora biti jedan iz grupe koju pripadni informacioni sistem prepoznaje. Ukoliko se ne definira entitet, informacioni će nabrojiti entitete koje u tom času može ponuditi za rad preko telexa.



UPIT

Lista entiteta:

Kontejner, putovanje broda, kurs din., tarife
////
(četiri kose crte označavaju kraj tekuće informacije)

Zatim slijedi postavljanje upita izborom ključnih pojmova po njihovim vrijednostima.

Opći oblik upita je:

UPIT/ENTITET/KLJUČ1=VRIJEDNOST/KLJUČ2=VRIJEDNOST

(NIVO 1) (NIVO 2) (NIVO 3)

- Ako na pojedinom nivou postoji samo jedan ključ, tada je dovoljno navesti vrijednost tog ključa
- Padom na niže nivoe, ključne vrijednosti viših nivoa se podrazumjevaju, te ih nije potrebno stalno navoditi.

No,uz poznavanje ključnih pojmova, moguće je direktno postaviti pitanje na bilo kom nivou (vidi primjer), odnosno doći do pravog podskupa podataka.

UPIT/KONT

Ključni pojmovi:

DEPO, STATUSI, BOOKING
////

DEPO

UPIT/KONT/DEPO

Slijedeći ključni pojmovi:

RJK (RIJEKA), KOPER (kombinirani sa vlasnikom JUL, LPR, SPP)
////

DEPO = JUL RJK

UPIT/KONT/DEPO = RJK DANA DD-MM-GG

STATUS	EMP	DAM	BOO	FEX	FIN	REP	REZ	STOCK
OTP -	164	12	-	122	31	4	44	100
DRY -	82	6	-	62	17	8	27	50
OST -	10	-	-	-	-	-	4	5

Slijedeći pojam - STATUS
////

STATUS = FIN

UPIT/KONT/DEPO=RJK/STATUS=FIN
(PUNI KONTEJNERI U UVOZU = FULL IMPORT)

BROD-PUTOVANJE:	STIGAO	IZ	LUKA	OTP	DRY
HRE 14/85	26.06.	NYC	BAL	20	10
JAX 7/85	27.06.	GOA		11	7

Slijedeći pojam - BROD-PUTOVANJE
////

BROPUT = HRE 14/85

UPIT/KONT/DEPO=RJK/STATUS=FIN/BROP=HRE 14/85

Preuzimaju

TRANSJUG za Tvornicu ... Zagreb, KEM... Skopje
JADROAGENT za Obuću ... Sarajevo
////
IZLAZ

KOMANDE EMULATORU

A. Privilegirane komande

TLX (CR) - startanje programa za dijalog

TLX) START - iniciranje početka veze

STOP - zaustavljanje komuniciranja ali tek nakon završetka trenutnog posla

PREKINI (CANCEL) - momentalno zaustavljanje komunikacije

KRAJ (END) - potpuni završetak rada sa izlaskom emulatora iz memorije. Pri tome emulator osigurava postojeću listu čekanja (spremanjem na disk) kako bi se ona kod slijedećeg startanja ponovo aktivirala.
Komande rade u parovima:
STOP+END ili CANCEL+END

Komande za definiranje načina rada informacijskog sistema u spoju sa telex mrežom.

TLX) PRIJEM - postavi prijem (SET MODE RECEIVE -ONLY)

TLX) PREDAJA - postavi predaju (SET MODE TRANSMIT-RECEIVE).

TLX) INTERAKT - postavi interaktivno (SET MODE (NO) INTERACTIVE)

TLX) NONINTER - briši interaktivno
- markira se dozvola za interaktivni pristup bazi

Napom.: Prethodnu grupu komandi ne može upotrijebiti bilo koji korisnik sistema, već samo za to posebno određeni korisnik - sa privilegijama operatera.

B. Neprivilegirane komande

TLX) LISTAJ - prikaz liste čekanja na terminalu sa svim podacima koji pripadaju listi.
a) broj pretplatnika koji treba pozvati
b) prioritet poziva
c) broj dosadašnjih pokušaja
d) broj uspostavljenih veza koje su nasilno (prije kraja) prekinute
e) naziv datoteke koja se šalje
f) redni broj člana (doznačava član koji je trenutno u transferu)

TLX) STATUS - prikaz stanja (za tekući dan)
a) broj članova u čekanju
b) broj poslanih telexa
c) broj primljenih telexa
d) broj ulaza u informacioni sistem putem ključne riječi
e) ukupno vrijeme rada
f) ukupno vrijeme rada na liniji

TLX) CALL/TLNC - xxxxx/tekst-ime 1, ime 2...,ime6
- ubacivanje tekstova u listu čekanja za specificirani telex broj

TLX) BRI/TLNO - xxxxx - briši sve pozive za taj broj

BRI/TEKST - ime - briši specificirani tekst iz svih mjesta u čekanju

TLX) MONITOR - vizuelna kontrola nad slanjem i primanjem (praćenje na terminalu)

TLX) IZLAZ (EXIT) - prekid dijaloga

ARHITEKTURA LOKALNE (MIKRO)RAČUNALNIŠKE MREŽE TI-30 ZA VODENJE PROCESOV

N.Panić, O.Mikulič, V.Kosmač, ISKRA-AVTOMATIKA, Ljubljana
Jugoslavija

LOCAL MICROCOMPUTER NETWORK ARCHITECTURE TI-30 FOR PROCESS CONTROL

UDK: 681.3.007

POVZETEK - Sodobne sisteme vodenja procesov lahko v celoti realiziramo v mikroračunalniški tehnologiji na podlagi decentralizacije funkcij vodenja v porazdeljenem računalniškem sistemu. Referat opisuje arhitekturo takšnega računalnika s poudarkom na sodelujoči avtonomiji fizičnih podsistemov, modularnosti, odpornosti na izpade, postopnem zmanjševanju funkcijske sposobnosti ob izpadih, razširljivosti in konfiguracijski prilagodljivosti.

ABSTRACT - Modern process control systems can be carried out effectively completely in microcomputer technology on the base of decentralizing of process control function in distributed computer system. The architecture of such computer system, with accent on subsystem, cooperative autonomy, modularity, fault tolerancy, graceful degradation capability, expansibility and configuration flexibility is described in this paper.

1. UVOD

Dobre lastnosti porazdeljenih sistemov vodenja so predvsem posledica decentralizacije funkcij vodenja na fizično ločene podsistome, za katere je značilna velika stopnja sodelujoče avtonomije. Tako lahko ekonomično realiziramo s pomočjo ustrezne porazdelitve funkcij vodenja v lokalni (mikro)računalniški mreži pomožna in glavna komandna mesta kot tudi fizično ločevanje med poljubno izbranimi nivoji interne hierarhije sistema vodenja (vertikalna modularnost) in fizično ločevanje med izbranimi funkcijskimi moduli znotraj izbranega hierarhijskega nivoja (horizontalna modularnost). Agregiranje in porazdeljevanje funkcij vodenja med večjim številom (mikro)računalnikov na podlagi formalno definiranega hierarhijskega funkcijskega modela omogoča maksimalno sodelujočo avtonomijo posameznih fizičnih podsistemov kot tudi minimalne podatkovne tokove med le-temi. (/3/,/6/).

Na področju praktične realizacije porazdeljenih sistemov vodenja prevladujejo sistemi, ki so še daleč od tega, da bi v polni meri dosegali zgoraj definirane lastnosti. Razlog je v tem, da imajo takšni sistemi samo decentralizirano in porazdeljeno ("inteligentno") periferijo (enote za zajemanje in predprocesiranje podatkov, regulatorji, programabilni krmilniki, enote za arhiviranje in protokoliranje, enote za povezavo človek-sistem) in še vedno preveč centralizirane globalne funkcije in obdelave (globalna podatkovna baza, koordinacijske in optimizacijske funkcije). "Inteligentna" periferija je običajno realizirana z mikroračunalniki in globalne funkcije in obdelave z zmogljivimi miniračunalniki. Vsi elementi sistema medsebojno komunicirajo preko lokalne mreže /1/,/2/.

Lahko ugotovimo, da realizacija praktično (ne samo pri nas, ampak tudi v svetu) zaostaja za sedanjo stopnjo razvoja mikroračunalniške tehnologije. Vzrok za to je v tem, da je zgradba "pravega" porazdeljenega sistema definirana v treh dimenzijah /5/:

- porazdeljenost aparaturne opreme
- porazdeljenost krmiljenja (operacijskega sistema)
- porazdeljenost podatkovne baze

in je potrebno obsežno, medsebojno tesno povezano raziskovalno delo v vseh treh dimenzijah. Pri tem je temeljno raziskovalno področje zgradba objektnega modela porazdeljenega sistema /4/, ki se prepleta, oz. je tesno povezan še z naslednjimi osnovnimi problemi:

- načini dostopa in porazdeljenost krmiljenja (nivo mreže,nivo operacijskega sistema, nivo podatkovne baze) in
- zmogljivost (efektivnost)

V tem smislu referat opisuje en pristop k gradnji osnovnega objektnega modela za realizacijo popolnoma decentraliziranih porazdeljenih sistemov, predvsem za vodenje procesov. Referat tudi opisuje osnovne principe realizacije tega objektnega modela s pomočjo lokalne mikroračunalniške mreže. Pri tem je podana tudi primerjava z drugimi znanimi rešitvami.

2. IZHODIŠČNI MODEL PORAZDELJENEGA RAČUNALNIŠKEGA SISTEMA

Vzemimo, da je porazdeljeni računalniški sistem fizično sestavljen iz N avtonomnih (centraliziranih) računalnikov oz. procesorjev, ki so medsebojno polno povezani preko nekega skupnega komunikacijskega medija.

Arhitekturo porazdeljenega sistema lahko obravnavamo na uporabniškem nivoju v dveh ločenih prostorih: logični in fizični. Logični prostor je kompakten uporabniški prostor porazdeljenega sistema, v katerem fizične meje med posameznimi procesorji niso vidne. V logičnem prostoru uporabniške naloge (tasks) medsebojno sodelujejo na podoben način kot da se izvršujejo na enem samem (super)procesorju. Druge, dodatne lastnosti logičnega prostora so določene z izbranim objektnim modelom. V fizičnem prostoru rešujemo problem določanja števila in porazdelitve uporabniških nalog med procesorji tako, da dosežemo naslednje lastnosti:

1) maksimalna sodelujoča avtonomija fizičnih podsistemov (procesorjev) oz. minimizacija informacijskega pretoka med le-temi;
2) določena stopnja odzivnosti sistema;
3) določena stopnja odpornosti na izpade (fault tolerance);
4) določena stopnja postopnega zmanjševanja funkcijske sposobnosti pri izpadih (grece full degradation capability) in
5) možnost postopne gradnje oz. naknadne razširljivosti brez vpliva na že zgrajene dele sistema. Pri tem postopku optimizacije je potrebno, da ima aparaturna oprema ustrezno modularnost in fleksibilnost , ki omogoča konfiguriranje posameznega procesorja v porazdeljenem sistemu, glede na dodeljene naloge. Obstaja medsebojni vpliv med lastnostmi porazdeljenega sistema v fizičnem prostoru. Najbolj je izražen vpliv lastnosti 1) na druge, eksistira pa tudi določen povratni vpliv. Bistveno je, da z doseganjem lastnosti 1) že v veliki meri dosežemo druge lastnosti /6/.

Za veliko stopnjo svobode glede gradnje in optimizacije porazdeljenega sistema v logičnem in fizičnem prostoru pri čim večji stopnji neodvisnosti med temi prostori (teoretični cilj je popolna neodvisnost), je skupnega pomena arhitektura objektnega modela.

3. OSNOVNI OBJEKTNI MODEL

Osnovni objektni model sestoji iz naslednjih osnovnih objektov /8/: naloge, virtualne komunikacijske enote, statusno polje porazdeljenega procesorja, realna ura in datum.

3.1. Naloge

Naloga je osnovni (nedeljivi) in edini osnovni aktivni objekt sistema. Naloge, ki medsebojno sodelujejo izključno preko virtualnih komunikacijskih enot, lahko poljubno porazdelimo med računalniki. Objektni model je posebej prilagojen k temu, da več porazdeljenih in/ali ponovljenih nalog sestavljajo logično in funkcijsko celoto - modul. Lastnost modula, da je lahko porazdeljen, je bistvenega pomena za gradnjo porazdeljenih globalnih resursov kot je distribuirana podatkovna baza.

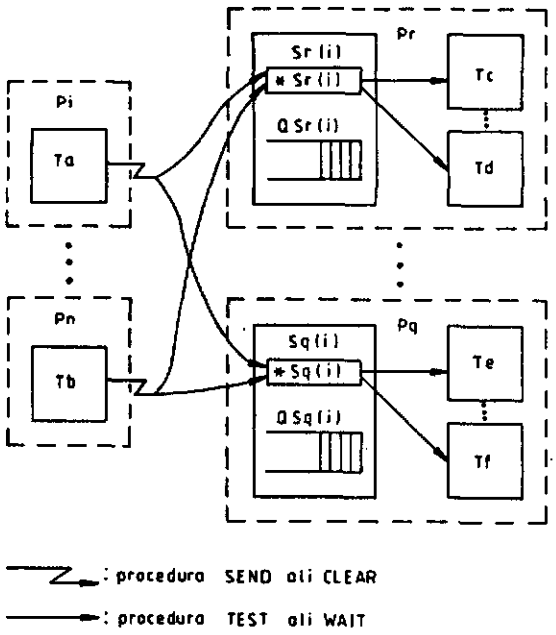
3.2. Virtualne komunikacijske enote

Virtualne komunikacijske enote omogočajo samo indirektno sodelovanje med porazdeljenimi nalogami. Te enote sestojijo iz logičnih signalov S(i) in logičnih kanalov CHj ; i,j=0,1.2.. ..,255.

Signali S(i) služijo za naslednje splošne namene: 1) vzpostavitev globalnih oz. transparentnih prednostnih in sinhronizacijskih razmer znotraj porazdeljenega sistema; 2) vzdrževanje globalnih statusnih informacij; 3) izmenjava enobitnih sporočil med nalogami in 4) zaščita lokalnih (znotraj posameznega procesorja) resursov.

Princip porazdeljevanja signalov in način medsebojnega sodelovanja med porazdeljenimi nalogami ilustrira sl. 1. Porazdeljevanje signalov med procesorji P_k se vrši po naslednjih pravilih:

- signal, ki je vhod v določeno nalogo, se vgrajuje obvezno skupaj s to nalogo v isti procesor
- določeni signal S(i) je lahko ponovljen v več procesorjih (ponovitve Sr(i), Sq(i) na sl. 1.), kar izhaja iz dejstva, da je določeni signal lahko vhod v več nalog in/ali ustrezna naloga je ponovljena v več procesorjih



Sl. 1.: Sodelovanje porazdeljenih nalog preko signalov

Posamezna ponovitev signala S(i) je lahko postavljena (*Sr(i)="1" ali zbrisana (*Sr(i)="Ø"); sl. 1. Opazovana ponovitev signala S(i) je lahko postavljena (monitorska procedura SEND) ali zbrisana (CLEAR) od strani več nalog iz istega in/ali drugih procesorjev. Po drugi strani nad posamezno ponovitvijo signala S(i) lahko več nalog izvrši proceduro TEST (branje) ali WAIT (čakanje na postavitev), vendar samo tiste naloge, ki so vgrajene skupaj z ustrezno ponovitvijo. Če je lokalna ponovitev signala v stanju "postavljena", procedura WAIT zbriše le-to in omogoči nadaljevanje izvršitve klicajoče naloge; drugače procedura postavi klicajočo nalogo v neaktivno stanje. Naloge, ki čakajo na postavitev določene ponovitve signala S(i) (npr. Sr(i) na sl. 1), so razvrščene v ustrezni vrsti QSr(i) glede na prioritete. Zbujevanje nalog iz vrste QSr(i) se vrši ob posamezni izvršitvi procedure SEND (S(i) nad Sr(i). Izvršitev procedure SEND (S(i)) ob prazni vrsti QSr(i), ima za rezultat postavitev Sr(i).

Vpliv procedure SEND in CLEAR na posamezno ponovitev Sr(i),..., Sq(i) opazovanega signala S(i) lahko nastavimo z izbiro enega izmed naslednjih statusov vgradnje:

- "strogo lokalna" (omogočen je vpliv samo iz smeri gostiteljskega procesorja, vsi zunanji vplivi iz/v gostiteljski procesor so onemogočeni)
- "lokalna" (vpliv je možen iz smeri gostiteljskega in drugih procesorjev, onemogočen vpliv iz smeri gostiteljskega procesorja na eventualne ponovitve v drugih procesorjih)
- "lokalna in globalna" (enako kot "lokalna" s tem, da je omogočen vpliv iz smeri gostiteljskega procesorja še na druge ponovitve)
- "globalna" (omogočen vpliv samo iz smeri gostiteljskega procesorja na ponovitve v drugih procesorjih). Ponovitev signala je globalna, če ni v enem izmed zgoraj navedenih stanj.

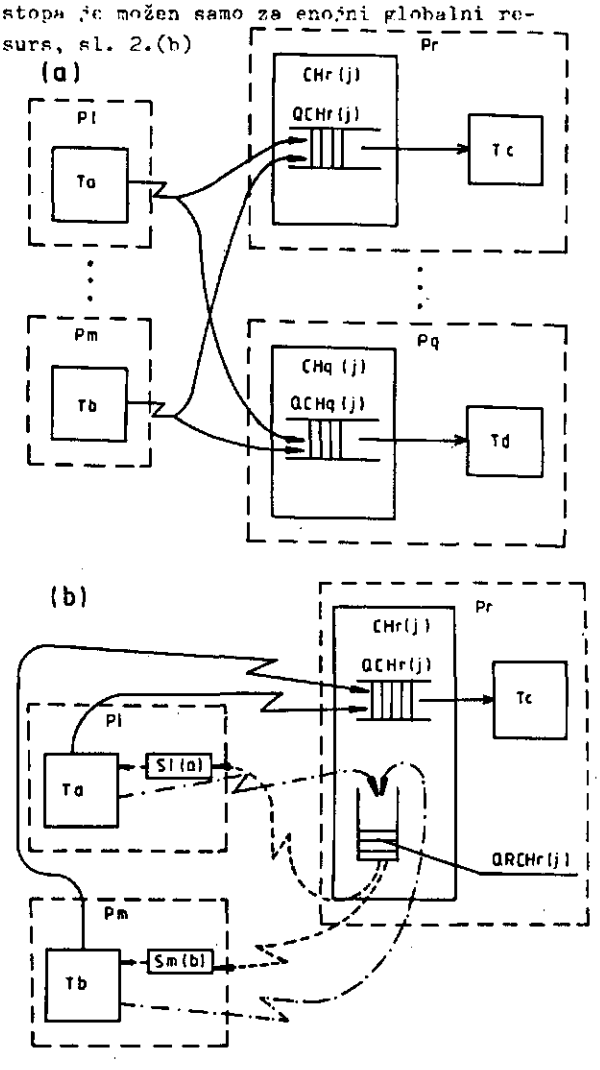
Statusi vgradnje omogočajo veliko fleksibilnost pri konfiguriranju porazdeljenega sistema. Tako lahko na enostaven način realiziramo ponovljen lokalni resurs z oznako S(i) na ta način, da vsem ponovitvam tega signala (Sr(i),...,Sq(i) damo status vgradnje "strogo lokalna" (ni prikazano na sl. 1). Vrste QSr(i),..... QSq(i) razrešujejo konflikte pri dostopu do ustreznih ponovitev lokalnega resursa. Pri tem lahko naloge konkurirajo medseboj neodvisno za zasedbo ustrezne ponovitve lokalnega resursa s pomočjo procedure WAIT (S(i). Sprostitev tega resursa naloga opravi s pomočjo procedure SEND(S(i)).

Kanali CH(j) služijo za naslednje splošne namene: a) vzpostavitev globalnih oz. transparentnih komunikacij med porazdeljenimi in eventuelno ponovljenimi nalogami in b) zaščita in realizacija globalnih, porazdeljenih ali emojnih resursov.

Naloge lahko medsebojno sodelujejo preko kanalov v splošnem na dva načina, sl. 2. Pri tem se porazdeljevanje kanalov med procesorji izvrši po enakih pravilih kot pri signalih. Posamezna ponovitev CHr(j),...,CHq(j) kanala CH(j) omogoča nabiranje uporabniških sporočil v ustreznih vrstah QCHr(j),...,QCHq(j). Te vrste so organizirane po pravilu "FIFO" in sprejemajo sporočila z omejeno maksimalno dolžino (do 255 zlogov). Nalaganje sporočil v posamezno ponovitev teh vrst lahko opravi medsebojno neodvisno več nalog iz poljubnega procesorja s pomočjo procedure SDATCH(CH(j)). Pri tem vpliv te procedure na posamezno ponovitev kanala CH(j) lahko nastavimo s pomočjo statusa vgraditve le-teh. Možni statusi vgraditve kanalov in pomen teh statusov je popolnoma enak kot pri signalih. Jemanje sporočil iz posamezne ponovitve vrste QCH(j) lahko opravi istočasno samo ena naloga s pomočjo procedure

RDATCH(CH(j)). Pri tem, takšna naloga in ponovitev kanala CH(j) morata biti vgrajeni v isti procesor.

Posamezni globalni resurs (disk, printer, podatkovna baza) ponazarja v logičnem prostoru porazdeljenega procesorja izbrani kanal. Pri tem določeni kanal CH(j) lahko realizira čisto transakcijski ali zaporedni (oz. "stream") način dostopa do globalnega resursa GR(j) = = CH(j). S pomočjo čisto transakcijskega načina dostopa lahko realiziramo globalni porazdeljeni resurs, sl. 2.(a). Zaporedni način dostopa je možen samo za enojni globalni resurs, sl. 2.(b)



Legendo (procedure):

RDATCH ; SDATCH ; REQ, REL

WAIT ; SEND

Sl.2: Sodelovanje med nalogami preko logičnih kanalov

a) porazdelitev nalog na istem kanalu možnost realizacije porazdeljenega globalnega resursa

b) realizacija enojnega (centraliziranega) globalnega resursa

Pravilo FIFO posamezne ponovljene vrste QCHr(j),...QCHq(j), razrešuje konflikte na nivoju posamezne transakcijske zahteve za CH(j), (sl. 2a). V primeru, da določena naloga (uporabnik) želi izvršitev neprekinjenega zaporedja lastnih transakcij, potem mora predhodno zasesti CH(j) za ekskluzivno uporabo (procedura REQ(CH(j)). Medsebojj neodvisne zahteve za ekskluzivno zasedbo kanala CH(j) se lahko nabirajo v vrsti QRCH(j), glede na prioritete sl. 2.b). Ustrezni nadzorovalnik vrste QRCH(j), ki je sestavni del CH(j), dodeljuje CH(j) k posamezni nalogi s pomočjo postavitve ustreznih povratnih signalov (signali Si(a) in Sm(b) na sl. 2). V ta namen sta definirana prioriteta naloge in indeks povratnega signala kot parametra procedure REQ. Ti parametri ponazarjajo posamezno nalogo v vrsti QRCH. Nad ekskluzivno pridobljenim resursom CH(j), naloga lahko opravi neomejeno število transakcij. Sprostitev resursa CH(j) naloga opravi s pomočjo procedure REL(CH(j)). Ta procedura obenem dodeli resurs CH(j) naslednji nalogi, ki eventualno čaka v vrsti QRCH(j).

S ciljem realizacije transparentne lastnosti logičnega prostora, porazdelitev signalov in kanalov, število ponovitev le-teh in statusi vgradnje so popolnoma skriti za naloge. Npr. neka naloga lahko postavi nek signal S(j) v proceduro SEND(S(j)), interna struktura objektnega modela pa preskrbi za ustrezno obdelavo vseh ponovitev Sr(j),...Sq(j) tega signala v procesorjih Pr,..., Pq. Po drugi strani opisani način porazdeljevanja signalov in kanalov, statusi vgradnje in funkcijski model le-teh omogoča veliko stopnjo svobode pri optimizaciji porazdeljenega procesorja v fizičnem prostoru (doseganje lastnosti 1) do 5) iz poglavja 2).

3.3. Sistemska ura in datum

S ciljem čimvečje avtonomije procesorjev, objektni model vzdržuje prenovljeno uro in datum v vseh procesorjih. Posebni interni protokoli znotraj objektnega modela skrbijo za vzdrževanje verne in sinhrone vrednosti le-teh.

3.4. Statusno polje porazdeljenega procesorja

Posebno polje logičnih signalov S(k), k=0,1,2, ...,N-1 realizira statusno polje porazdeljenega procesorja na naslednji način: če procesor P_{k+1} deluje in je priključen v komunikacijsko omrežje porazdeljenega procesorja, potem *S(k)=Ø. Statusno polje služi predvsem za realizacijo funkcij maskiranja okvar in rekonfiguriranja sistema v primeru le-teh.

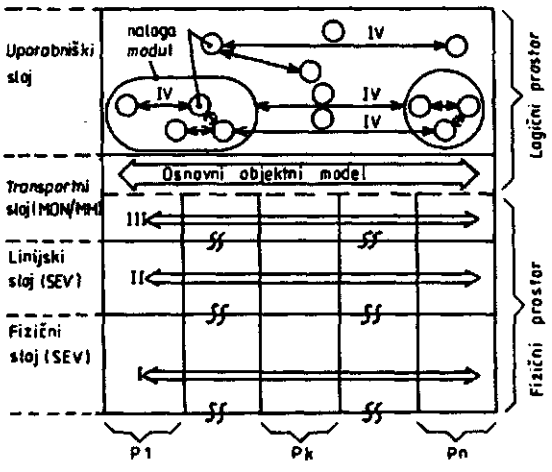
lahko služi tudi v globalne diagnostične namene in protokoliranje okvar. S ciljem doseganja maksimalne avtonomije procesorjev, je statusno polje ponovljeno v vseh procesorjih. Interna struktura objektnega modela skrbi za vzdrževanje medsebojno vernega stanja teh ponovitev.

3.5. Diagnostika

Okvare in/ali logične napake na uporabniškem nivoju lahko pripeljejo v porazdeljenem sistemu do mrtvih objemov in/ali neskončnih blokiranj. Opisani objektni model ne vsebuje funkcij izogibanja, ampak funkcije odkrivanja in razreševanja teh stanj. Te funkcije zajemajo: 1) sprostitev čakajočih vrst po preteku časa, ki je definiran od strani prizadetih nalog; 2) vrnitev programske kontrole, skupaj s statusom napake, v prizadete naloge; 3) decentralizirano procesiranje napak na posameznem procesorju od strani posamezne prizadete naloge in/ali od strani posebnega sistemskega nadzorovalnika (poskusi "ozdravitve", rekonfiguriranje) in 4) decentralizirano generiranje diagnostičnih sporočil za centralizirani prikaz, alarmiranje in protokoliranje.

4. REALIZACIJA PORAZDELJENEGA RAČUNALNIKA

Model porazdeljenega računalnika iz poglavja 2 in 3 lahko realiziramo s slojevito arhitekturo, ki je maksimalno prilagojena na opisani objektni model, sl. 3. V tej arhitekturi je število slojev minimizirano v primerjavi z ISO modelom /9/. Razdeljeni računalnik s slike 3. ne vsebuje slojev mreže, seje in prezentacije. Problem prezentacije je prepuščen uporabnikom. Funkcije seje so integrirane v transportni sloj. Vzdrževanje dialogov med uporabniki je prepuščeno uporabnikom v logičnem prostoru. Na podlagi opisanega objektnega modela lahko realiziramo v tem prostoru, znotraj uporabniškega sloja, dodatne sistemske sloje (sloj resursov), kot tudi poljubne uporabniške funkcijske sloje (glej funkcijski model sistema vodenja) iz /6:. Mrežni sloj je nepotreben, ker se adresiranje vozlišč mreže (tukaj označenih kot procesorji P_1..., P_k...P_N) lahko vrši indirektno (asociativno) preko adresiranja elementov objektnega modela. Tukaj imajo odločilno vlogo statusi vgraditve signalov in kanalov, ki so skriti kot konfiguracijski podatki v transportnem in linijskem sloju. Princip te komunikacije lahko pojasnimo z naslednjim primerom: če neka naloga sproži izvršitev procedure SEND(S(i)) v procesorju Pe, potem transportni in linijski sloj poskrbita za to,



I : fizični protokol — III : transportni protokol

II : linijski protokol — IV : uporabniški protokol

Sl. 3: Arhitektura porazdeljenega računalnika TI-30

da se operacija SEND opravi na vseh dostopnih ponovitvah signala S(i) glede na statuse vgraditve tega signala v posameznem procesorju. Ta koncept lahko obdržimo tudi v primeru več povezanih mrež z ustrezno uporabo statusov vgraditve signalov in kanalov v povezovalnih vratih (gateways).

Posledica maksimalne prilagoditve celotne zgradbe porazdeljenega računalnika na opisani objektni model je posebni protokol znotraj linijskega sloja (linijski protokol) /10/. Ta protokol se bistveno razlikuje od znanih in standardnih protokolov (HDLC, Ethernet, IEEE 802) predvsem v naslednjem:

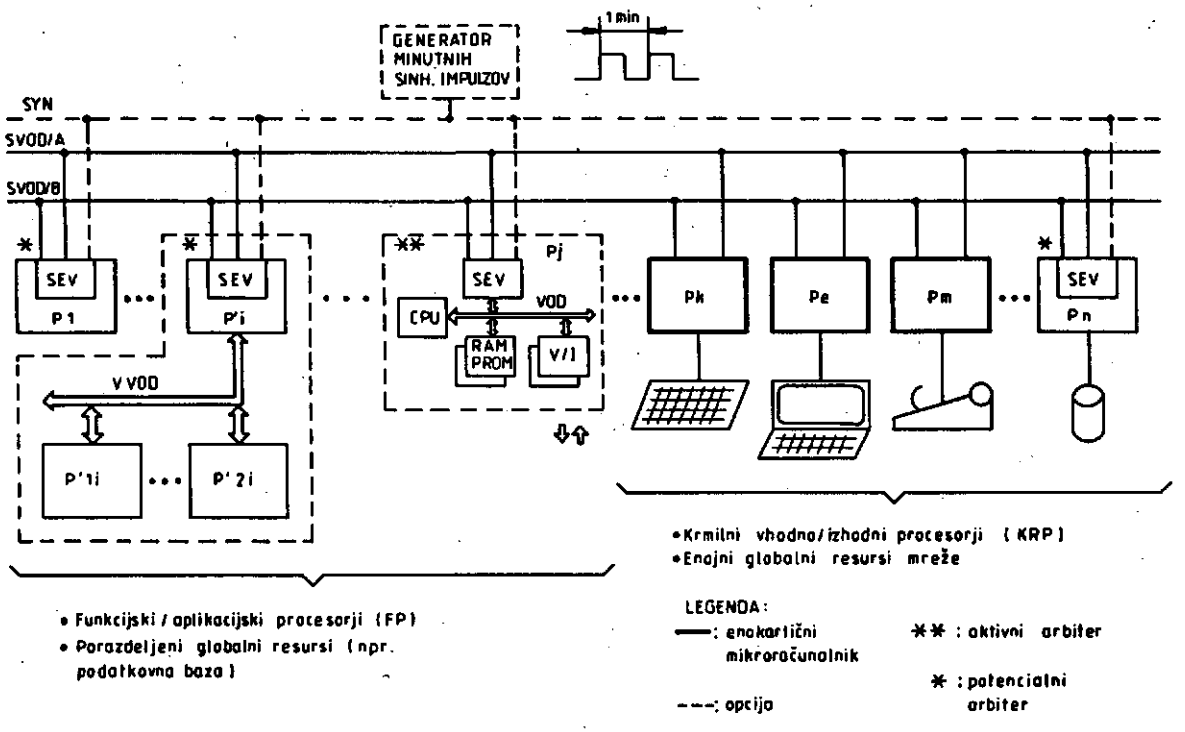
1. Linijski sloj aktivno sodeluje v funkcijah vzdrževanja objektnega modela na uporabniškem nivoju (sistemski čas in datum, statusno polje lokalne mreže). V konvencionalnih protokolih je linijski sloj v celoti pasiven proti nadrejenim slojem (deluje kot strežnik).
2. Elementi objektnega modela (logični kanali in signali) so vključeni v linijski protokol kot ciljne (logične) adrese sporočil. Razen teh adres ni drugih ciljnih adres. V konvencionalnih protokolih je objektni model na uporabniškem nivoju popolnoma skrit na nivoju linijskega protokola. Ciljne adrese sporočil pa označujejo adreso fizičnega vozlišča (računalnika) v mreži.

3. Linijski protokol je optimiziran za komunikacije "en oddajnik - več sprejemnikov" (posledica koncepta ponovljenih objektov oz. porazdeljenih modulov na uporabniškem nivoju). Kontrola pretoka in zaščita proti izgubi sporočil je realizirana z negativnim kvitiranjem. Konvencionalni linijski protokoli so optimizirani za komunikacije "en oddajnik - en sprejemnik". Kontrola pretoka in zaščita proti izgubi sporočil je realizirana s pozitivnim kvitiranjem.

4. Kot posledica lastnosti pod 3., imajo vodilno vlogo pri krmiljenju prenosa sporočil na linijskem nivoju sprejemniki, v konvencionalnih protokolih pa oddajniki.

Transportni sloj porazdeljenega računalnika je realiziran z multimikroračunalniškim monitorjem MON/MM, tako da se le-ta vgradi v posamezni procesor /8/. Pri tem je v posamezni izvedbi MON/MM skrita konfiguracijska tabela statusov vgraditve signalov in kanalov v pripadajočem procesorju. Linijski sloj posameznega procesorja P je realiziran z enokartičnim mikroračunalnikom SEV. SEV sodeluje z gostiteljskim procesorjem oz. z ustrezno izvedbo, MON/MM preko skupnega pomnilnika,(ki je v gostitelju). SEV omogoča realizacijo porazdeljenega računalnika v obliki lokalne mreže, sl. 4.

Lokalna mreža TI-30/10/ sestoji iz določenega števila (maksimalno 64) (mikro)računalnikov oz. procesorjev P_1, P_e, ... P_N, ki so organizirani okrog podvojenega serijskega vodila SVOD/A in SVOD/B, sl. 1 Procesorji P so principielno razdeljeni v dve skupini: funkcijski procesorji FP in krmilni procesorji KRP, FP-i so namenjeni za realizacijo porazdeljenih globalnih resursov in za izvrševanje vnaprej dodeljenih uporabniških funkcij (npr. določenih funkcij vodenja)/6/. KRP-i so namenjeni za krmiljenje določenih perifernih enot in za povezavo le-teh v lokalno mrežo kot enojnih globalnih resursov. Za realizacijo bolj enostavnih enojnih globalnih resursov (printerji, terminali) so procesorji tipa KRP lahko v celoti realizirani kot enokartični procesorji. Splošno, posamezen P lahko sestoji iz več hierarhijsko bolj tesno povezanih mikroračunalnikov. Taka skupina je organizirana okrog "vertikalnega" paralelnega vodila VVOD, kjer je glavni računalnik povezan na skupno serijsko vodilo. Posamezni P lahko podvojimo v celoti ali samo glavni del le-tega z namenom povečane zmogljivosti. Podvojeni P-i delujejo kot "on-line/stand-by" konfiguracije. Razen s skupnim serijskim vodilom SVOD/A,B je lokalna mreža lahko povezana še s sinhronizacijsko linijo,



Sl. 4: Splošna konfiguracija lokalne mikroračunalniške mreže TI-30

ki služi za prenos minutnih sinhroimpulzov v posamezni P-r iz nekega zunanjega referenčnega generatorja. Za medsebojno sinhronizacijo sistemskih ur v lokalni mreži, ta mikrolinija ni nujno potrebna: prenos minutnih mikrosporočil je sestavni del komunikacijskega protokola na SVOD/A,B.

Dodeljevanje skupnih vodil SVOD/A,B se vrši s pomočjo posebej prirejene metode podajanja žetonov (token passing) oz. z realizacijo logičnega prstana. Posebnost uporabljene metode dodeljevanja skupnega vodila v primerjavi s standardnimi (npr. IEEE 802.4) je predvsem v naslednjem: 1) podajanje žetonov vrši arbiter, 2)žeton služi, razen za dodelitev vodila določenemu P-u, tudi kot pridružena adresa izvora v eventualno oddanem sporočilu (izvor je adresa procesorja, ki je oddal sporočilo) in 3) arbiter lahko s pomočjo žetona preklopi komunikacijo iz SVOD/A na SVOD/B in obratno. S pomočjo koncepta arbitra bistveno poenostavimo problem reševanja globalnih funkcij linijskega sloja lokalne mreže. Obenem s tem, da ima žeton dodatne pridružene funkcije, poenostavimo linijski protokol in povečamo izkoristek le-tega. Iz razlogov zanesljivosti lahko funkcije arbitra izvršuje več SEV-ov (potencialno vsi), pri čemer je vedno samo eden aktiven in drugi pripravljeni za delovanje. V primerih okvare aktivnega arbitra eden izmed pasivnih in pripravljenih arbitrov avtomatsko prevzame aktivno vlogo. V danem trenutku kompletna komunikacija poteka samo po enem SVOD/A ali SVOD/B. Arbiter opravi preklop komunikacije v primeru predolge zasedbe vodila od strani P-a (posamezni P-r lahko odda ob dodelitvi SVOD samo eno sporočilo oz. blok dolžine do 255 zlogov) v primeru, da poklicani P-r ne prevzame vodila.

5. SKLEP

Večina znanih porazdeljenih sistemov temelji na objektnih modelih. Edine potencialne slabosti takšnih modelov so /4/: relativno manjše časovne performanse (zaradi neprilagojenosti celotne zgradbe sistema izbranemu objektnemu modelu) in granulacija objektov (problem določanja optimalne velikosti objekta v smislu informacijskega in funkcijskega obsega). Prva pomanjkljivost praktično ni prisotna v sistemu TI-30, ker je celotna arhitektura porazdeljenega procesorja maksimalno prilagojena izbranemu objektnemu modelu. Druga pomanjkljivost je zmanjšana na minimum pri definiciji osnovnih (nedeljivih) objektov.

Opisana arhitektura ustreza "teoretični" definiciji porazdeljenega računalnika iz /5/ in obenem kaže na glavne vzroke, zaradi katerih konvencionalni porazdeljeni sistemi ne morejo zadostiti tej definiciji. Glavni vzrok je v tem, da so porazdeljeni sistemi bili predmet raziskav in razvoja predvsem z vidika komunikacijskega omrežja in ne z vidika porazdeljenega računalnika.

Opisana arhitektura porazdeljenega računalnika je uporabna za realizacijo poljubnih porazdeljenih sistemov za delovanje v realnem času, posebej pa je prilagojena potrebam realizacije maksimalno decentraliziranih porazdeljenih sistemov vodenja procesov. V tem smislu smo definirali logični in fizični prostor porazdeljenega sistema. S tem, da smo ustvarili transparentni logični prostor, smo dosegli lastnosti, ki jih imajo v tem pogledu centralizirani sistemi (velika modularnost in konfiguracijska fleksibilnost programske opreme). Po drugi strani, ne da bi porušili razmere v logičnem prostoru, lahko decentraliziramo in optimiziramo porazdeljeni sistem v fizičnem prostoru z veliko stopnjo svobode. Tako lahko na ekonomičen način dosežemo lastnosti 1) do 5) iz poglavja 2. Iste lastnosti v večini centraliziranih sistemov vodenja je nemogoče doseči ali pa je potrebno za to plačati nesprejemljivo visoko ceno.

LITERATURA:

/1/ J.M.Ayache,J.P.Courtiat and M.Diaz,"REBUS", A Fault-Tolerant Distributed System for industrial Real-Time Control", IEEE Trans. on Computers, vol. C.-31,No.7, July 1982, pp 637 + 647

/2/ M.S.Sloman,S.Prince, "Local Network Architecture for Process Control", Proc. of the IFIP Working Group 6.4 International Work shop on Local Networks, Zürich 27 + 29 Aug. 1980, pp 407 + 427

/3/ H.Ihara and K.Mori, "Autonomous Decentralized Computer Control Systems", Computer, Vol. 17, No 8, August 1984 pp 57 + 66

/4/ J.A.Stankovic, "A perspective on Distributed Computer Systems", IEEE Trans. on Computers, vol. C-33, No.12, December 1984, pp 1102 + 1115

/5/ P.H.Enslow, "What is a "Distributed" Data Processing System?, Computer vol 11, No.1 January 1978, pp 13-21

/6/ M.Panić, L.Lenart, P.Peterlin, O.Mikulič, A.Uratnik, J.Škrabe, J.Kanduč, J.Petelinkar "Multimikroračunalniške izvedbe kompleksnih sistemov vodenja procesov v realnem času", Informatica 2/3, 1983, str. 121 ÷ 128

/7/ M.Panić, L.Lenart, O.Mikulič, M.Kikelj, V.Kosmač "Programska oprema distribuiranih sistemov vodenja procesov", Informatica 2/3, 1983, str. 127 ÷ 132

/8/"Multimikroračunalniški monitor MON/MM, navodilo za uporabo", Iskra-Avtomatika TOZD Razvojni inštitut, Interna dokumentacija

/9/ ISO/TC97/SC16/N227 "Reference Model of Open System Interconnection (Version 4)", August 1979

/10/ "Opis lokalnih mikroračunalniških mrež TI-30, ISKRA-Avtomatika, TOZD Razvojni inštitut, Interna dokumentacija

UPORABA PRI IZOBRAŽEVANJU

USAGE IN EDUCATION

PRIMJENA KOMPJUTERA U NASTAVI NA PREHRAMBENO-BIOTEHNOLOŠKOM FAKULTETU SVEUČILIŠTA U ZAGREBU

Želimir Kurtanjek
Prehrambeno-biotehnološki fakultet
Zagreb , Jugoslavija

UDK: 681.3.014

U radu je izložena primjena kompjutera u nastavi na Prehrambeno-biotehnološkom fakultetu Sveučilišta u Zagrebu. Ukratko je izložen nastavni plan u okviru predmeta iz matematike, automatizacije, simuliranja procesa i operacijskog istraživanja. Opisana je primjena minikompjutera kao najpovoljniji oblik samostalnog rada studenata. U nastavi je naglašen sistemski pristup opisu procesa. Svaki proces je prikazan kao bilanca mase , energije i količine gibanja na osnovu kojih je izvode matematički modeli. Istaknut je značaj djelovanja informacija na ostvarivanje svrhovitosti sistema. Pojedina poglavlja iz matematike obrađena su u obliku algoritama koji su prilagođeni za jednostavnu primjenu kod sastavljanja simulacijskih programa za minikompjuter.

APPLICATION OF COMPUTERS IN TEACHING AT THE FACULTY OF FOOD AND BIOTECHNOLOGY , THE UNIVERSITY OF ZAGREB

In this work is described application of computers in teaching at the Faculty of Food and Biotechnology at The University of Zagreb. The teaching program of computer application is given as parts of lectures in mathematics , process control, system simulation and operational research. The use of small computers has been found to be the most convinient way for individual work of students in groups in laboratories. In lectures is given emphasis on system approach for process modelling and analysis. Each process is described as balances of energy , mass, and momentum, by which is derived a mathematical model. The importance of information interaction of a subject with system is stressed, because it is a factor which determines performance of the whole system. Some parts of lectures in mathematics are presented as algorithms which can be adapted for simple application in simulation programs on small computers.

Uvod

U izobrazbi kadrova iz biokemijskog i prehrambenog inženjerstva na Prehrambeno-biotehnološkom fakultetu, Sveučilišta u Zagrebu, znatna pažnja se posvećuje uptrebi kompjutera. Inženjerski pristup rješavanja problema zahtjeva da budući inženjeri od početka studija prihvate primjenu elektroničkog računala i metode informatike kao osnovna sredstva u radu. Budući da je studij koncepiran na eksperimentalnim metodama, to je primjena kompjutera naročito izražena pri planiranju eksperimenata i statističkoj obradi podataka. Stečena znanja omogućuju bolje razumjevanje složenih biokemijskih i fizičkih procesa, eksperimentalni podaci se interpretiraju s statističkom značajnošću, a planiranje eksperimenata skračuje rad u laboratoriju i pojednostavnjuje analizu. Razumjevanje tehnoloških procesa je znatno unapređeno metodama simulacije na računalu i primjeni postupaka analize iz automatskog vođenja procesa.

Nastavni program

Znanja iz osnova programiranja stječu se u prvom semestru prve godine studija u okviru predmeta iz matematike. Za programiranje je predviđeno 15 sati predavanja i posebne vježbe sa praktičnim radom na računalu. Studenti se upoznavaju s osnovama programskog jezika BASIC i jednostavnim numeričkim algoritmima.Obrađuje se rješavanje linearnih jednadžbi, Newton-Raphsonov algoritam za nelinearnu jednadžbu, i jednostavne metode integracije. Svaki student dobiva za svoj zadatak koji samostalno rješava i polaže ga u obliku seminarskog zadatka. Seminar je zamjena za prijašnji kolokvij iz logaritamskog računala (šibera). Rješenje zadatka se predaje u obliku programa koji se mora izvesti na računalu. Stečena znanja se nadograđuju kroz rad na predmetima viših godina. Studenti primjenjuju kompjuter na predmetima Fizikalna kemija, Tehnološke operacije, Mjerenja, regulacija i automatizacija, Matematičko medeliranje i vođenje industrijskih procesa i Operacijsko istraživanje. Udio rada na komjuteru se mijenja u pojedinim predmetima, tako da je veća zastupljenost u predmetima koji imaju naglasak na matematičkom modeliranju i upotrebi matematičkih algoritama. Primjenjuju se metode koje su prikladne za analizu složenih procesa, simulaciju dinamike , i algoritmi operacijskog istraživanja. Većina numeričkih

metoda je napisana u obliku [illegible] se studenti upoznavaju samo s matematičkim osnovama i uputama za korišćenje odgovarajuće subrutine. Takovi programi su na primjer : Gausl za sistem algebarskih jednadžbi, Newton-Raphsonov alogoritam za sistem jednadžbi, matrične operacije , Runge-Kutta 4, Simplex algoritam , metoda najmanjih kvadrata, metoda kolokacija. Studenti također koriste gotove programe u koje upisuju svoje podatke, ili sastavljaju glavni program koji nadopunjuju s potprogramima. Većina programa je pisana u BASIC-u i sastavljeni su tako da se mogu vrlo jednostavno modificirati i primjeniti za neki drugi problem. Zadaci koje studenti obrađuju obuhvaćaju probleme kao što su statistička analiza eksperimentalnih podataka, projektiranje jednostavnijih tehnoloških operacija, analiza kinetike biokemijskih reakcija, analiza vođenja procesa , optimalno komponiranje prehrambenih proizvoda itd. Od programskih jezika osim BASIC-a studentima se ukazuje na osnove FORTRAN-a i PASCAL-a. U okviru predavanja iz automatizacije studenti se upoznaju s strukturom računala i osnovama programiranja mikroprocesora. Kroz predavanja se obrađuju osnovni elementi i načini povezivanja procesa i elektroničkog računala u svrhu praćenja i vođenja procesa. Praktični rad se izvodi na mikroračunalima tipa SPECTRUM, TEXAS INSTRUMENTS. ORAO-VELEBIT i također postoji mogućnost rada u Sveučilišnom računskom centru SRCE. Osnovna djelovanja mikroprocesora upoznavaju se kroz rad s mikroračunalom ISKRA DATA UMRS1 (MOTOROLA 6800).

Nastava iz simuliranja i vođenja procesa

Sistemski pristup i metodike primjene kompjutera se izlažu kroz predmete Mjerenje, regulacija i automatizacija i Matematičko modeliranje i vođenje industrijskih procesa. Prvi predmet se predaje na trećoj godini i obavezan je za studente biokemijskog i prehrambenog smjera, dok je drugi predmet na četvrtoj godini i obavezan je samo za studente biokemijskog inženjerstva. Sistemski pristup se uvodi u dijelu predmeta gdje se određuju opće značajke mjernih uređaja. Svaki mjerni uređaj se analizira kao odnos mjernog signala i mjerene veličine. Točnija definicija sistema se izlaže u uvodu u automatsko vođenje procesa. Iz mjerenja je obrađeno područje primjene kompjutera za statističku obradu podataka. Studenti obrađuju rezultate dobivene u laboratoriju na kompjuteru primjenjujući statističke testove, interva. prave vrijednosti,izračunavanje korelacija i procjenjuju parametre. Metoda najmanjih kvadrata se koristi za obradu podataka dobivenih kalibracijom mjernih osjetila i uređaja. Sistem se definira odnosom ulaznih veličina, veličinama stanja procesa , i izlaznim veličinama [1-5]. Struktura svrhovitog sistema se uvijek prikazuje kao cjelina procesa i jedinice za vođenje. Interakcija sistema i okoline opisuje se kao djelovanje okoline koje je određeno ulaznim procesnim veličinama i ulaznim informacijskim veličinama. Procesne ulazne veličine određuju prijenos mase, energije i količine gibanja između okoline i sistema, i djeluju isključivo na proces. Ulazne informacijske veličine djeluju isključivo na jedinicu za vođenje. Analiza sistema na kompjuteru svodi se na određivanje matematičkog modela procesa , algoritmiranja djelovanja jedinice za vođenje i primjene matematičkih metoda. Naglasak u nastavi je na principima izvođenja matematičkih modela. Model procesa se uvijek izvodi na osnovu diferencijalnih bilanci koje definiraju jednadžbe za veličine stanja procesa. Izlazne veličine se izvode iz stanja procesa i određene su svrhom sistema. Numeričke metode i programi za primjenu kompjutera se dijele kao i sami modeli na one s koncentriranim parametrima i distribuiranim. Simulacija jedinice za vođenje ograničena [illegible] jednostavne dvopoložajne regulatore i PID regulatore. Struktura koja povezuje pojedine veličine procesa i jedinice za vođenje se uvijek analizira kao kombinacija triju osnovnih oblika , programnog vođenja, unaprijednog i u povratnoj vezi.
Nastava iz modeliranja i vođenja industrijskih procesa karakterizirana je specifičnostima pojedinih inženjerskih problema. Kompjuterski programi su znatno složeniji, više dimenzionalni i zahtjevaju veći broj podataka. Prije izvođenja vježbi na računalu,studenti se upućuju na pretraživanje literature radi određivanja vrijednosti karakterističnih tehnoloških podataka i parametara specifičnih za pojedini proces. Problemi koji se analiziraju najčešće odgovaraju realnim industrijskim procesima. Rade se primjeri [6-9] šaržne fermentacije, reaktori s imobiliziranim enzimima, cijevni reaktori, mikrobiološki aktivni filmovi, smrzavanje prehrambenih proizvoda, sušenje u fluidiziranom sloju, komponiranje hrane , filtracije, destilacija itd.

Zaključak

Upotreba informatike i kompjutera je od izuzetne važnosti u naobrazbi inženjerskog kadra iz biokemijskog i prehrambenog inženjerstva. Elektronička računala i numeričke metode trebaju se uvesti na prvoj godini studija, tako da se primjena unapređuje kroz specifičnosti pojedinih stručnih predmeta na višim godinama. Dosadašnja iskustva pokazuju da je primjena jednostavnih mikroračunala pogodna za rad s studentima u grupama , budući da se ne zahtjeva skupi centralni sistem a broj obrađenih programa može biti dovoljno velik u vrijeme predviđeno za studentske vježbe. Isto tako održavanje mikroračunala je vrlo jednostavno i ne zahtjeva neke posebne troškove. Relativno jednostavno se može izgraditi programoteka

numeričkih metoda , statisističkih testova i datoteka osnovnih podataka koje mogu u velikom dijelu zadovoljiti edukativne potrebe.
Nakon savladavanja metodike primjene kompjutera za vrijeme studija, budući inženjer se mnogo lakše uključuje u interdisciplinarne timove i u rad s profesionalnim softwerom u većim računskim centrima.

Literatura

1. G. Stephanopolous, Chemical Process Control, Prentice-Hall International Series, Englewood Cliffs, New Jersy, 1984.
2. W.L. Lyben , Process Modelling , Simulation , and Control for Chemical Engineers, Mc. Graw Hill, New York , 1973.
3. D.R. Coughanower, L.B. Koppel, Process Systems Analysis and Control, Mc Graw Hill, New York, 1965.
4. V.V. Kafarov , Kibernetika u kemiji i kemijskoj tehnologiji, Tehnička knjiga, Zagreb, 1970.
5. T. Šurina , Automatska regulacija, Školska knjiga, Zagreb, 1981
6. H.A. Leniger , W.A. Beverloo, Food Process Engineering, D.Reide Publishing Company, Boston 1975.
7. S.E. Charm , The Fundamentals of Food Engineering, AVI Publishing Company Inc., Westport, Connecticuit , 1979.
8. J.E. Bailey, D.F. Ollis, Biochemical Engineering Fundamentals , Mc Graw Hill Book Company, New York, 1977
9. B. Atkinson , Biochemical Reactors, Pion Limited, London, 1974.

THE POTENTIAL OF MICROCOMPUTER GRAPHICS IN THE TEACHING OF SOME CLASSES OF COMPRESSIBLE FLUID FLOW PROBLEMS

M. A. Brebner

Dept. of Computer Science
University of Calgary
CALGARY
Alberta T2N 1N4
CANADA

UDK: 681.3.06

Abstract

Using a good BASIC, FORTRAN or PASCAL compiler, it is possible to obtain the numerical solution of some compressible fluid flow problems in one space variable and time, in the order of one to five minutes on an IBM PC with an Intel 8087 mathematical coprocessor. In the case of IBM BASICA or TURBO PASCAL, the graphics extensions of these languages allow the easy development of pictorial output, which aids the teaching of compressible fluid flow.

Another advantage in the case of the IBM BASICA is to use the interpreter version's ability to temporarily interrupt the computation and display or print the current values contained in selected variables. A further option is the print screen key on the IBM PC, which allows the graphics displayed on the screen to be dumped to a suitable dot matrix printer.

If an IBM PC with 64Kb memory, graphics adapter card, graphics monitor and at least one 360Kb diskette drive is available at the conference some examples can be demonstrated as part of the presentation.

Introduction

Using a good BASIC, FORTRAN or PASCAL compiler, it is possible to obtain the numerical solution of some compressible fluid flow problems in one space variable and time, in the order of one to five minutes on an IBM PC with an Intel 8087 mathematical coprocessor [see 1,2]. In the case of IBM BASICA or TURBO PASCAL, the graphics extensions of these compilers allow the programmer to easily display plots of the dependent variables, such as pressure, against the space or time variable as the computation proceeds. This capability has great potential in the teaching of a topic as complex as compressible fluid flow.

Another advantage in the case of IBM BASICA, is that a student (or a teacher demonstrating a concept in a class) using the interpreter version, can temporarily interrupt the execution of the program, and use BASIC in immediate mode to print single values or whole arrays of values. This dump of values can be to the screen or, using LPRINT, to the printer. The advantage of routing the output to the printer is that the graphics displayed on the screen will not be disturbed. In fact, even more powerful processes can be invoked. The student having studied the output values, can instruct the computer to resume the computation by depressing a single key.

A further option that is available on the IBM PC is the print screen key, which allows the graphics displayed on the screen to be dumped to a graphics printer if the appropriate DOS option has been set. It is therefore possible for the student to stop the execution of a program at various times and obtain "snapshots" of the graphics displayed on the screen. Sufficient graphics capability for this feature is now being incorporated in many of the lower price microcomputer dot matrix printers. With more effort it is also possible to arrange for a compiler version of the program to be interrupted to allow a graphics dump, if the appropriate compiler options are set for interactive interrupts.

Compressible Fluid Flow Program

A program has been developed which models compressible fluid flow in one space variable and time. With minor modifications the program can simulate various conditions, such as:

(i) A boundary being rigidly fixed or moving at a varying speed.

(ii) A fixed or variable pressure condition at a boundary.

(iii) An internal discontinuity, such as an interface between two fluids or a shock wave.

To illustrate these different types of conditions modification can be made by the teacher. Alternatively the students can be asked to carry out this task as a course exercise.

The Eulerian and Lagrangian forms of the mathematical equations and finite difference schemes for the compressible fluid flow problem are described in [2]. In the program a perfect gas law with gamma, the ratio of specific heats, given the value 3, and the Lagrangian form of the equations were used. The initial pressure and density were set as 300 and 1 respectively. These were selected to simplify checking some of the results.

Four types of problem have been successfully run. These are:

(a) One boundary moving with a smoothly varying, sinusoidal speed of small amplitude and a fixed boundary.

(b) One boundary moving under the force of a varying external pressure and the other boundary fixed.

(c) One boundary instantaneously being set in motion with a finite fixed or varying speed, which forms an instant shock wave travelling ahead of the moving boundary and the other boundary fixed.

(d) Both boundaries moving in various combinations of the above cases.

Some typical examples of the print screen output given at the end of the paper, are discussed in the following sections.

Sinusoidal Velocity Boundary Conditions

The program was run with one and both boundaries

moving in a sinusoidal manner. For the examples shown the boundary condition used is given by

$$U = +2*(1-COS(4*PI*T)),$$

where U is the left hand boundary velocity and T is time. Diagrams 1 show plots of pressure against a set of 100 equally spaced Lagrangian coordinates for the problem with the above boundary condition.

With reference to diagrams 1(a) and (b), note the way the front of the waves rise more steeply as they progress to the right. This is due to the compressible nature of the fluid, which produces variations in sound (wave propagation) speeds. In diagram 1(c), the first wave has been reflected from the rigid (zero velocity) right hand boundary. Subsequently in diagrams 1(d) and (e), the waves travelling in both directions interact to form peaks, troughs and plateaux. It should be noted the generation of these diagrams requires only simple techniques, that could easily be used by students while they are experimenting with various models. Also, of course, a dynamic sequence of almost continuously varying graphics could be presented to the students, by producing screen output every time step. This dynamic presentation greatly enhances the student's comprehension of the subject being taught.

Piston/Shock Problem and Dumps of Variables

In the piston/shock problem [see 2,3], the left hand boundary moves into the fluid with a velocity of 10, forming a shock wave moving with a velocity 41.623, and causing a jump in the pressure and other dependent variables. Later the shock wave meets the right hand boundary which is held rigid (velocity held at zero), and the shock wave is reflected with a second jump in pressure, etc., "behind" the shock wave, which is now travelling from right to left. Yet later, the shock wave meets the moving piston (left hand boundary) and is reflected a second time with a further jump in the dependent variables "behind" it. These processes continue to be repeated as time increases.

In diagrams 2(a) and (b), the movement of a set of equally spaced particles in the fluid are shown, by plotting the Eulerian coordinates of the selected particles at each time step. The shock waves progress is clearly shown by the changes in the direction of particle paths as their velocities alternate between 10 and 0.

Diagrams 3 show an unusual combination of pressure plotted against the Eulerian coordinates of the same set of particle paths used in diagrams 2. These are plotted for every time step, giving an envelope of pressures, which tend to smooth out the pressure profiles.

Because, in a certain sense, this problem is a relatively simple case, it is possible to solve the Rankine-Hugoniot equations for conditions "behind" the shock front, for each successive reflection. The theoretical calculated pressure and velocity "behind" the original shock wave (a), and the first (b) and second (c) reflected shock waves, are given below, and can be compared with the numerical results in diagrams 3(a), (b) and (c).

	Pressure	Velocity
(a)	716.23	10.0
(b)	1395.70	0.0
(c)	2401.67	10.0

Because of the numerical methods used for smoothing out the discontinuities, the values printed oscillate about the correct results, and within a few mesh points of the shock front there is a transition in the size of the values shown.

The line of statements which prints a slice of the one dimensional arrays P (pressure) or U (velocity) need only be typed in once. When the program is again interrupted by depressing the control-break keys, the user only has to position the cursor level with the line of statements using the arrow keys, and then press return to obtain the required output. The only danger is printing over the graphics already displayed on the screen, which causes the overprinted regions to be lost. The pressure and velocity slices are displayed on the graphics dumps, although in the case of the velocity, only the values are given to save repeating exactly the same graphics. These dumps could have been sent directly to the printer using LPRINT instead of PRINT. It should be noted that the arrow and delete keys can be used to "clean up" the screen, but of course there is no way to simply recover parts of the screen that have been overwritten.

More complex tasks, if they are anticipated, can be carried out by including in the program suitable subroutines, which can be invoked by typing GOSUB followed by the appropriate line number. These features can all be used with interpreter version of BASIC. In the case of compiled programs some of these options are not available, but the print screen key will usually function when the program is interrupted in a suitable manner. There are two obvious ways to achieve an interruption of a compiled program, (i) by the program requesting input from the keyboard at an appropriate place in the program, and at a time tested for in the program, or (ii) by setting certain compiler options to allow special interrupt keys to be tested by the program.

Varying Velocity Boundary Conditions

This option has to be used with some care, as an acceleration followed by a deceleration may cause cavitation (the pressure will become negative in the region of cavitation), which is not allowed for in the model incorporated into the program.

An example with a varying velocity boundary condition has already been given in the section and diagrams for the sinusoidal velocity condition.

Conclusion

The static diagrams presented in this paper only give an indication of the graphics capabilities of microcomputers. The full potential of the dynamic graphics interaction possible can only be shown by running suitable demonstration programs on a microcomputer, such as the IBM PC.

Acknowledgements

The author would like to acknowledge the assistance of the Computer Applications Unit, Faculty of Education, University of Calgary for providing both the software and hardware used for this paper. In particular, I would like to thank Ann Brebner, Herbert Hallworth, Heber Jones, and Dagmar Walker.

REFERENCES.

1. Brebner, M. A., Benchmarking Microcomputers for Some Mathematical and Scientific Computations, Infomatica 85, Nova Gorica, Sept. 1985.

2. Brebner, M. A., Modeling Fluid Dynamic Shock Wave Problems on a Microcomputer, Society for Computer Simulation Modeling and Simulation on Microcomputers Conference, San Diego, Jan, 1985.

3. Courant, R. and Friedrichs, K. O., Supersonic Flow and Shock Waves, Interscience,1948.

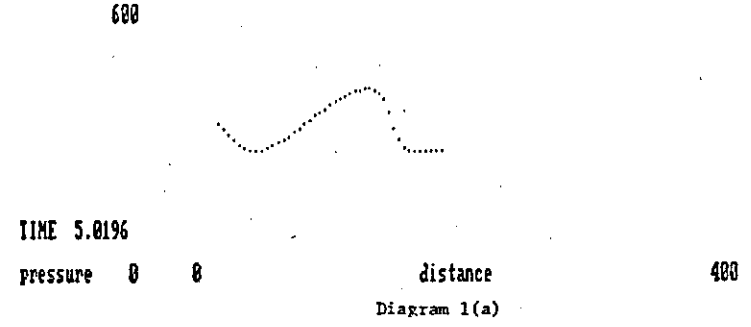


Diagram 1(a)

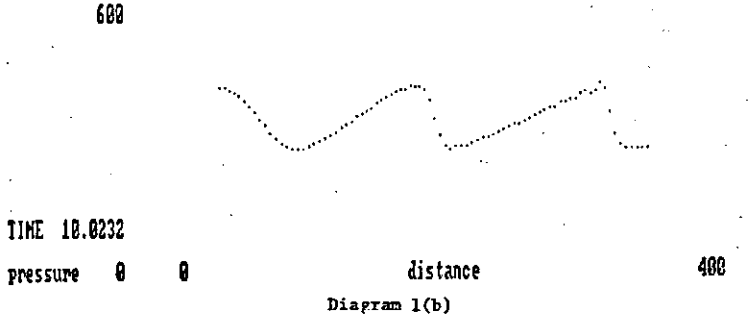


Diagram 1(b)

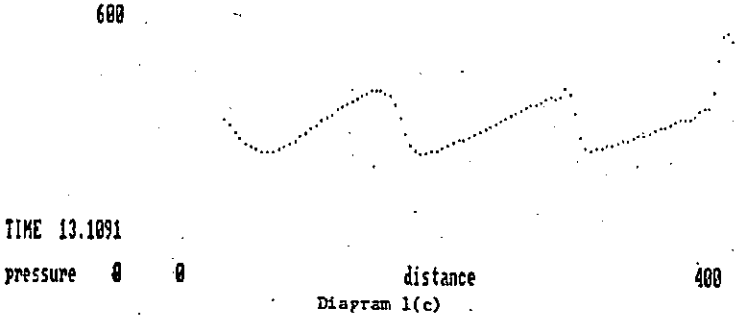


Diagram 1(c)

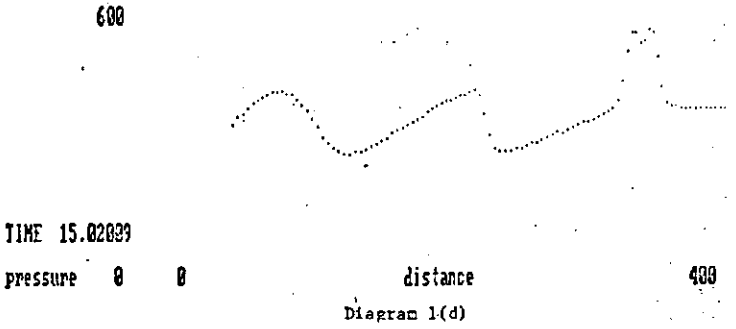


Diagram 1(d)

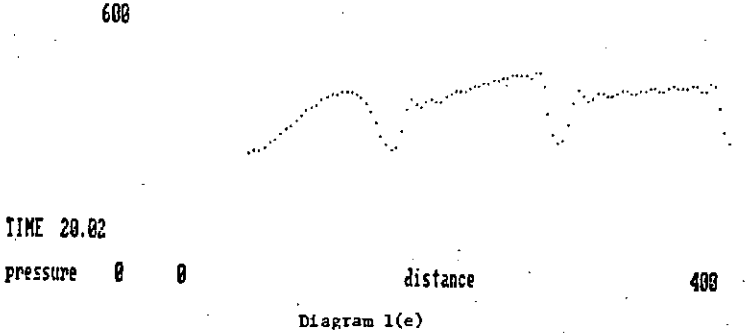


Diagram 1(e)

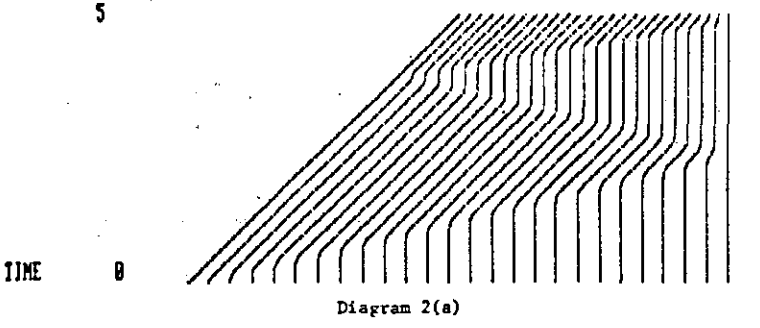


Diagram 2(a)

```
 6000
TIME  .9999994
^C
Break in 524
Ok
INPUT J,K:FOR L=J TO K:PRINT P(L),:NEXT L
? 1,17
 715.4846      717.9135      711.7586      723.9832      704.4788
 726.5028      712.321       700.8371      742.425       636.3913
 449.1891      335.1622      303.9339      300.2447      300.012
 300           300
Ok
 3000

INPUT J,K:FOR L=J TO K:PRINT U(L),:NEXT L
? 1,17
 10.0318       9.941884      10.07912      9.954797      9.9092
 10.25528      9.565891      10.28518      9.965475      6.867882
 2.748596      .4374942      3.882471E-02                1.631628E-03
 5.937865E-05                1.64381E-11   2.502149E-25
```

Diagram 3(a)


```
 6000
TIME  2.839998
^C
Break in 412
Ok
INPUT J,K:FOR L=J TO K:PRINT P(L),:NEXT L
? 9,25
 714.7969      719.58        711.3887      721.5743      712.4358
 716.4959      722.3936      716.4513      775.5888      901.3938
 1146.166      1367.575      1416.987      1373.261      1390.334
 1411.85       1382.737
Ok
 3000
INPUT J,K:FOR L=J TO K:PRINT U(L),:NEXT L
? 9,25
 10.06873      9.950818      10.00652      10.05545      9.883877
 10.1305       9.839303      9.616842      8.245924      4.73585
 1.531669     -.0673544     -.1083055      .1340632      6.364923E-03
-3.910895E-02                0
```

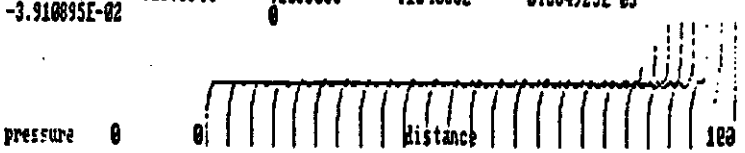


Diagram 3(b)

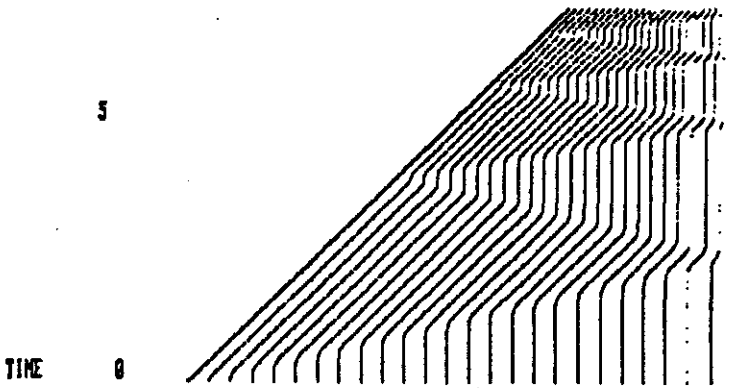
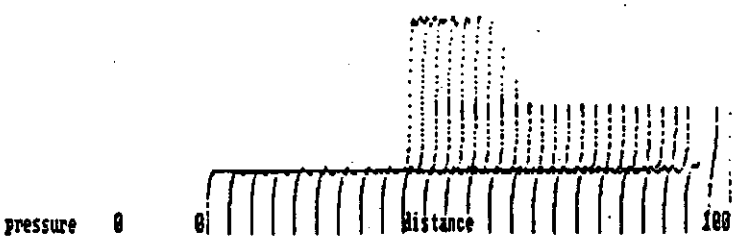


Diagram 2(b)

```
 6000
TIME  4.190001
^C
Break in 524
Ok
INPUT J,K:FOR L=J TO K:PRINT P(L),:NEXT L
? 1,17
 2396.035      2406.879      2413.76       2377.481      2376.579
 2431.943      2363.546      2066.458      1706.075      1475.658
 1405.089      1399.393      1398.808      1392.452      1394.034
 1400.015      1394.446
Ok
 3000
```




```
INPUT J,K:FOR L=J TO K:PRINT U(L),:NEXT L
? 1,17
 9.896451      10.12595      10.15632      9.660621      9.961614
 10.23434      8.615468      5.867818      1.730522      .4194192
 7.068349E-02                -4.220756E-02               -6.709044E-03
 4.491174E-02                -.0257661     -.0264843     3.150005E-02
```

Diagram 3(c)

NEKE PRIMENE APPLE II MIKRORAČUNARA U NASTAVI FIZIKE

Srboljub Stamenković
Sava Maksimović
Prirodno matematički fakultet
Institut za fiziku
Kragujevac, Jugoslavija

UDK: 681.3.06

U radu je dat kratak pregled aktivnosti Laboratorije za računske mašine i programiranje u Institutu za fiziku PMF u Kragujevcu. Ukazano je na tri vrste programa koji se prave u toj laboratoriji, kao i na to da se najveća pažnja posvećuje programima za primenu računara u nastavi fizike.

SOME APPLICATIONS OF APLLE II MICROCOMPUTER IN THE TEACHING OF PHYSICS: Presented in the paper is a short overview of the activities of the Laboratory for Computing Machinery and Programming in the Physics Institute of the Faculty of Natural Sciences in Kragujevac. Three kinds of programs currently underway in the Laboratory are discussed, the greatest attention being devoted to the programs for computer aided teaching of physics.

1. UVOD

Pre izvesnog vremena Institut za fiziku PMF u Kragujevcu nabavio je dva mikroračunarska sistema APPLE II, što je omogućilo formiranje Laboratorije za računske mašine i programiranje. Zadatak ove laboratorije je bio da omogući što širu primenu pomenutih mikroračunara u nastavi i studentskim radovima u okviru Instituta, a po mogućstvu i šire. Pored neophodne obuke studenata, sadržaj rada ove laboratorije bio je i stvaranje programa, najpre kraćih a potom i onih ambicioznijih. O toj vrsti rada biće nešto više rečeno u ovom radu.

2. VRSTE PROGRAMA

Aktivnost saradnika Laboratorije za računske mašine i programiranje bila je veoma bogata i raznovrsna pa su iz nje proizašli veoma raznovrsni programi. Ipak, svi oni se mogu razvrstati prema svojoj nameni u tri grupe:

a. Programi za demonstraciju u nastavi fizike
b. Programi za obradu podataka i proračun
c. Programi za individualnu nastavu (samoučenje)

Najveći deo programa napisan je u BASIC-u, nešto manji broj u PILOT-u, a za složene grafičke ambicije korišćen je mašinski jezik.

3. PROGRAMI ZA DEMONSTRACIJU U NASTAVI FIZIKE

U prvu grupu spadaju oni programi čija je namena da nastavu fizike učine očiglednijom, interesantnijom, potpunijom, a time i efikasnijom. Ovi programi su prvi koji su napravljeni u pomenutoj laboratoriji i njihova primena predstavlja pripremnu fazu u procesu stvaranja računske nastave. Korišćenje programa za demonstraciju ima za cilj da poboljša tradicionalni vid nastave, a ne da je podredi računaru. Računar je samo moćno sredstvo u nastavi, a ne zamena nastavnika. Iz ove grupe navodimo kao primer kratak prikaz demonstracije kosog hica. Nastavnik na uobičajen način izlaže gradivo, a računar koristi povremeno kada treba da nacrtra trajektoriju ovog kretanja, da nadje maksimalni domet i visinu, dajući mu odredjene podatke. Za odredjene uslove treba zadati početnu brzinu, elevacioni ugao i masu. Na monitoru se iscrtava tačka po tačka trajektorije. Unošenjem podataka za korekciju zbog otpora vazduha dobija se i balistička kriva-realna putanja kod kosog hica. Po završenom izlaganju nastavnik slušaocima još nekoliko puta ponovi demonstraciju davanjem različitih podataka za početne uslove.

Pored ovoga uradjeno je još niz sličnih programa kao: "X-zraci i njihova primena", "Doplerov efekat", "Konstrukcija lika svetle tačke kod sočiva", itd.
Program "Konstrukcija lika svetle tačke kod sočiva" je nešto širi i sastoji se od nekoliko kraćih programa medjusobno povezanih. Svaki od tih manjih programa obradjuje odredjenu fazu procesa konstrukcije lika, a može se "startovati", odnosno izvršavati izdvojen iz celine.

4. PROGRAMI ZA OBRADU PODATAKA I PRORAČUN

Ovi programi za sada su naišli na najširu primenu, posebno kod studenata i asistenata. Naročito su popularni programi za obradu podataka dobijenih u laboratorijskim vežbama, za nalaže-

nje raznih vidova devijacija i crtanje dijagrama. Jedan od takvih malih, a studentima veoma korisnih programa je i "Metoda najmanjih kvadrata".

Studenti, naročito oni iz nižih semestara, pri izradi laboratorijskih vežbi, najčešće se suočavaju sa problemom crtanja grafika linearne funkcije ili onih koje mogu da se svedu na linearne. Takav je slučaj kod studentskih vežbi: "Odredjivanje ubrzanja Zemljine teže matematičkim klatnom", "Nalaženje modula elastičnosti i modula torzije jedne žice", "Provera Omovog zakona u kolu jednosmerne struje". Pri starom uobičajenom načinu računanja neophodnih parametara za crtanje dijagrama metodom najmanjih kvadrata, gubi se mnogo vremena, pa često sporiji studenti nisu stizali da vežbu u potpunosti završe. Korišćenjem programa "Metoda najmanjih kvadrata" problem se rešava za par minuta. Na zahtev računara korisnik daje (ukucava preko tastature) vrednosti nezavisno promenljive veličine, zavisno promenljive, kao i broj vrednosti. Računar obradjuje podatke i na monitoru ispisuje srednju vrednost merenja kao i odstupanje pojedinih merenja od srednje vrednosti. Najzad ispisuje vrednosti koeficijenata (a) i (b), potrebnih za crtanje dijagrama i traži razmeru za crtanje. Kada mu se da vrednost razmere računar crta pravu liniju odredjenog koeficijenta pravca i odredjenog odsečka na ordinatnoj osi.

U ovu grupu spadaju i "Programi za proračun" koji se koriste u izradi diplomskih i magistarskih radova, kao i u naučno istraživačkim projektima. Jedan od takvih je i "UGR", koji modeluje ugao rasejanja fotona u Komptonovom efektu Von Neumanovom metodom. Ili "FLUKS" koji služi za izračunavanje ekspozicione doze u zatvorenim prostorima po metodi "Monte Karlo" (2).

5. PROGRAMI ZA INDIVIDUALNU NASTAVU

Treću grupu programa čine oni namenjeni individualnoj nastavi. Toj vrsti programa se u ovom trenutku posvećuje najveća pažnja, i u toku je pokušaj da se sačini jedan širi programirani materijal koji bi obuhvatao čitav opšti kurs fizike za studente mašinskog fakulteta.

Deo posla je obavljen. Iz odeljka mehanika, oscilacije i toplota obradjeno je po petnaest, a iz optike i nuklearne fizike po deset lekcija. Iako nijedna sekvenca nije sasvim konačna, na njima se stalno vrši dogradnja i poboljšanje, mogu se uočiti konture konačnog oblika programiranog materijala. U ovoj početnoj fazi nemamo pretenzije da ovaj program u potpunosti zameni nastavnika fizike u toku celog semestra, već samo u onim delovima ovog kursa fizike koji su naročito pogodni za programiranje i realizovanje pomoću mikroračunarskog sistema APPLE II. Znači nastavu bi "zajednički" izvodili nastavnik i računar, a podela posla bi bila izvršena na najcelishodniji način: odredjeni broj lekcija (one koje posebno odgovaraju) studentima bi prezentirao računar, odredjen broj sam nastavnik (lekcije ne pogodne za programiranje) dok bi odredjeni broj lekcija nastavnik izlagao koristeći računar samo kao pomoćno sredstvo za demonstracije. Jedan od programa ove vrste je i program za učenje oblasti fizike "Radioaktivnost i primena radionuklida". Taj program sačinjen je za potrebe eksperimentalne nastave u jednom odeljenju usmerenog obrazovanja, "Kragujevačke gimnazije". Program je u toku drugog tromesečja (krajem 1984. godine) prezentiran i učenici su ga izvanredno primili. Sastoji se od osam sekvenci koje odgovaraju osam metodskih jedinica u tradicionalnom vidu nastave. Svaka sekvenca sadrži nekoliko članaka, a svaki članak jezgrovitu informaciju odredjenog nastavnog sadržaja, zadatak kojim se proverava da li je informacija usvojena i najzad rešenje zadatka. Korisnik (učenik) koji zadatak ispravno reši obaveštava se o tome i upućuje na sledeći članak, odnosno sekvencu. Korisnik koji da pogrešno rešenje ili odgovor, dobija dopunsku informaciju ili se upućuje na ponovno učenje, a potom pokušava da reši zadatak.

Testovi vršeni u toku ovog eksperimenta ukazuju na niz neospornih i u literaturi ukazanih prednosti nastave uz pomoć računara.

6. ZAKLJUČAK

I pored odredjenih teškoća zbog ograničenih mogućnosti mikroračunarskog sistema APPLE II sa jednom disk jedinicom, tokom poslednje dve godine koliko postoji i radi Laboratorija za računske mašine i programiranje, postignuti su značajni uspesi u raznim vidovima obrazovnog rada. Obzirom da se očekuje nabavka jednog moćnijeg mikroračunarskog sistema, optimizam u pogledu daljeg napretka u razvoju obrazovanja uz pomoć računara u našoj sredini je sasvim opravdan.

LITERATURA

1. A. Stamatović: "Primena mikroprocesora i mikroračunara u obrazovnom radu u okviru studija fizike", IBND.151, 1982.
2. D.Nikezić, Magistarski rad PMF,Kragujevac, 1984.
3. P. Mandić, Inovacije u nastavi i njihov pedagoški smisao,"Svjetlost", Sarajevo, 1977.

PRIPRAVA VZGOJNO-IZOBRAŽEVALNIH MIKRORAČUNALNIŠKIH PROGRAMOV IN NJIHOVA UPORABA PRI POUKU GEOGRAFIJE

Tatjana Ogrinc, Bibijana Mihevc
INŠTITUT ZA GEOGRAFIJO UNIVERZE EDVARDA KARDELJA
Ljubljana, Jugoslavija

Ob izdelavi vzgojno-izobraževalnih mikroračunalniških programov moramo upoštevati vrsto zahtev: od pravilne izbire snovi, do izbire načina dela (spoznavanje snovi, ponavljanje, preverjanje znanja, simulacije, iskanje informacij) ter upoštevanja značilnosti stroke, v našem primeru geografije. Mikroračunalniški programi Hidroenergetske osnove Jugoslavije nas seznanjajo s problematiko pridobivanja hidroenergije v Jugoslaviji, preverjajo pridobljeno znanje in so zasnovani kot programirane sekvence. Predstavljajo prvi primer predstavitve geografske snovi s pomočjo mikroračunalnika.

PREPARATION OF EDUCATIONAL MICROCOMPUTER'S PROGRAMMES AND THEIR APLICATION BY GEOGRAPHICAL LESSONS

Preparing educational programmes for microcomputer we have to take in account several special demands: first of all the right choise of topic and clearly defined way of use (recognition of the topic, repetition, test of knowledge, simulation, searching of informations) as regards special framework of specific disciplines, in this case characteristics of geography. Microcomputer programmes "Hydroenergetic potencials of Yugoslavia" give us possibility to introduce questions of producing hydroenergy in Yugoslavia, to check gained knowledge and they are planned as programmed sequences. These programmes are the first attempt in using microcomputer for educational purpose in geography for elementary and high schools in Slovenia.

Računalniško opismenjevanje je zajelo velik del šoloobvezne mladine. Vrsta osnovnih šol je nabavila mikroračunalniško opremo in v okviru raznih krožkov animirala mlade s tečaji programiranja. Sedaj se pojavlja nova ovira - pomanjkanje programske opreme, primerne za vzgojno-izobraževalno delo. Na vrzel v opremljanju z domačo programsko opremo je opozorila Zveza organizacij za tehnično kulturo Slovenije in predlagala izdelavo vrste mikroračunalniških programov, ki smo jih pričeli pripravljati tudi na Inštitutu za geografijo Univerze Edvarda Kardelja.

Pri delu smo naleteli na vrsto problemov, od katerih naj navedemo pravilno izbiro tematike, ki bi bila najustreznejše predstavljiva s pomočjo računalnika.

Drug problem predstavljajo osnovne zahteve geografske stroke in sicer: kompleksno gledanje na geografske pojave v njihovi medsebojni povezanosti in vzajemnih vplivih, prikaz geografskih elementov in pojavov v prostoru in času ter ugotavljanje procesov sprememb in zakonitosti njihovega razvoja. Vse to bi moralo biti upoštevano, poleg splošnih didaktičnih in metodičnih načel, pri oblikovanju vsebinske plati programa za pouk geografije.

Geografija je eden izmed predmetov, pri katerem je za njegovo proučevanje uporaba mikroračunalnika še toliko bolj privlačna in primerna. Npr. pri pouku velikokrat uporabljamo najrazličnejše podatke, zmanjkuje pa časa, da bi te podatke analizirali in poiskali njihove medsebojne povezave in zakonitosti. Če pa nam računalnik nudi tako bazo podatkov, ki jo z lahkoto in predvsem zelo hitro uporabimo za razne izračune in grafične predstavitve, je to lahko eden od zelo uspešnih učnih pripomočkov. Take baze podatkov bi lahko učenci pod vodstvom učitelja pripravljali tudi sami ali pa jih vsaj dopolnjevali.

Kot učni pripomoček računalnik omogoča prehod od obravnave dejstev in pomnenja množice podatkov k reševanju problemov, ki zahtevajo ustvarjalno razmišljanje, spodbujajo radovednost in kreativno sposobnost otrok.

V geografiji bi bili poleg tega zelo uporabni programi, ki simulirajo določene pojave tako, da učenec odloča, računalnik pa prikaže posledice odločanja. Pri tem verjetno ni potrebno posebej poudarjati, da ima tak način učenja mnogo večji učinek, če lahko vsak učenec sam odloča ob svojem računalniškem kompletu, kot pa če sta v razredu le 2 računalnika. V okviru geografske stroke je bilo narejenih že nekaj poiskusov prenosa učne snovi na računalnik. Eden izmed teh je kaseta Hidroenergetske osnove Jugoslavije, s petimi programi, pripravljenimi za računalnik Sinclair Spectrum 48 k.

V okviru učnega načrta za 8.razred osnovne šole, smo se odločili za predstavitev aktualne problematike - energije. Programi so kot dopolnilo uporabni tudi pri pouku v šolah. Glede na osnovne geografske zahteve smo skušali učence seznaniti z naravnimi elementi, med drugimi z rekami, povodji ter rečnimi režimi, ki pomenijo pokazatelje značilnosti reke in so rezultat klimatskih, reliefnih značilnosti, vegetacije in drugih elementov. Kažejo na kolebanje vodnega stanja preko

leta in vplivajo na energetsko izkoriščenost reke. Od naravnih elementov smo v programu prešli na energetsko izkoriščanje rek po vodno gospodarskih območjih, seznanili smo učence s pomembnejšimi hidroelektrarnami v Jugoslaviji, z njihovo močjo, v zadnjem delu smo predstavili proizvodnjo električne energije po letu 1960 ter strukturo pridobljene energije glede na poglavitne energetske vire.

Pri pripravi programov smo se trudili, da ne bi zanemarili načela nazornosti, zato smo pripravili več skic, kartogramov in grafikonov, npr.: povodja v Jugoslaviji s pripadajočimi rekami, reke z lokacijami hidroelektrarn, vodostaje rek, proizvodnjo električne energije in podobno. Programi so sestavljeni tako, da omogočajo spoznavanje učne snovi postopno ali naenkrat, omogočajo poljubno število ponavljanj posameznih učnih enot in stalno preverjanje pridobljenega znanja v testih, ki sledijo posameznim enotam. Programe lahko uporabljamo za seznanjanje učencev z novo snovjo, lahko preverjamo pridobljeno ali dopolnjujemo osnovno znanje.

Gre torej za programe, ki so vsebinsko zasnovani podobno, kot programirane sekvence, ki so predvsem primerne za individualno delo, saj omogočajo hiter tempo dela in so primerne za učence z različnim predznanjem, prispevajo pa k znatno večji aktiviranosti in k razvijanju samostojnosti pri delu.

Verjetno pa v bodoče ne bo poudarek na tovrstnih programih, podobnih programiranim sekvencam, ampak na takih, ki še v večji meri omogočajo nov pristop in nov način spoznavanja učne snovi. Poudarek bo verjetno na bazah podatkov, na simulacijah, na reševanju problemov, ki silijo k logičnemu razmišljanju in kritičnemu vrednotenju podatkov in rezultatov.

Pri uvajanju in uporabi mikroračunalniške opreme v šolah pa moramo opozoriti še na en problem - izobraževanja učiteljev. Pomen učitelja v procesu vzgoje in izobraževanja ob uporabi računalnika raste, saj je učitelj pomemben vodnik pri reševanju tovrstnih problemov. Povsem neprimerna in neopravičljiva je trditev, da bodo računalniki zamenjali učitelja. Računalniki lahko le pripomorejo k boljši kvaliteti pouka, zato ne smemo pozabiti na izobraževanje in primerno motiviranost učiteljev, od katerih bo, potem ko bodo programi že na trgu, odvisno koliko in kako jih bodo vključili v vzgojno-izobraževalni proces.

UMETNA INTILIGENCA

ARTIFICIAL INTELLIGENCE

A FUNCTIONALLY DISTRIBUTED ARCHITECTURE FOR THE IR-NLI EXPERT INTERFACE

Giorgio Brajnik, Giovanni Guida
and Carlo Tasso

Istituto di Matematica, Informatica e Sistemistica
Università di Udine
Italy

UDK: 681.3:159.953

ABSTRACT

A datailed analysis and evaluation of the current capabilities and limitations of the IR-NLI expert interface are presented. Its main architecture is described and the specific control mechanism adopted, called task, is further discussed. A description of its main limitations follows, and the requirements of a novel more advanced architecture based on a distributed approach are illustrated and briefly compared with other proposals.

1. Introduction

The IR-NLI project [5] concerns the development of an expert interface capable of modelling the intermediary to an online information retrieval system. The first version of the system is written in Franz LISP on a SUN-2 workstation and relies on a rule-based system which implements the core REASONING MODULE. In this module explicit use of meta-knowledge has been made through the task mechanism, which is able to isolate functionally heterogeneous competences at various levels of abstraction. In such a way different aspects of the intermediary's activity can be explicitly encoded and taken into account. Going further along this line, in this paper we consider the more advanced proposal of organizing the REASONING MODULE as a distributed problem solver where each submodule has definite roles and where precise cooperation mechanisms are specified. The paper is organized as follows: section 2 illustrates the basic organization of the IR-NLI system; section 3 focuses on the capabilities and limitations of the task mechanism; section 4 introduces a new proposal of a cooperative functionally distributed architecture as an alternative to the task mechanism; section 5 discusses major points of future research.

2. Basic Organization of IR-NLI

It is well known that the user of an information retrieval system often prefers to rely on the assistance of an expert professional (the intermediary) capable of effectively carrying out the search. The interaction between the user and the intermediary starts with a presearch interview aimed at clarifying content and objectives of user's needs. The intermediary chooses the most appropriate data base to access and devises a suitable search strategy, i.e. a formal program expressed in the query language of the data base. The search strategy is then submitted to the information retrieval system in order to select the items relevant to the initial user's request. The design of the search strategy includes a very critical activity: the selection of appropriate terms, and the identification of the relationships among them that can adequately represent the user's needs and are suitable for accessing the relevant records in the data base.

The main goal of the IR-NLI system is to model the intermediary's behaviour. The overall architecture of the system is shown in Figure 1 and it comprises three main modules:

- UNDERSTANDING AND DIALOGUE MODULE, devoted to perform activities of linguistic nature. It first translates the natural language user's request into a formal internal representation (PIR), which is later expanded during the presearch interview. The UNDERSTANDING AND DIALOGUE MODULE utilizes a vocabulary and a data base of linguistic knowledge. In the current implementation of the system, the UNDERSTANDING AND DIALOGUE MODULE has not been implemented, and it is manually simulated.
- REASONING MODULE, devoted to devise suitable search strategies. It is also responsible for the overall control of the activities performed by the system.
- FORMALIZER MODULE, devoted to translate the final completely expanded PIR produced by the REASONING MODULE, into the syntactic form of the query language of the data base currently accessed.

As already pointed out, the REASONING MODULE is the kernel of the whole system. It has been implemented as a rule-based system utilizing a base of expert knowledge, concerning the evaluation of user's request, the control of the presearch interview, the expansion of the PIR, the selection of a suitable approach (among the several possible ones currently utilized by human intermediaries) for the

design and generation of the search strategy, the execution of all the lower level actions that implement the chosen approach, and the possible engagement of the user in a clarification dialogue.

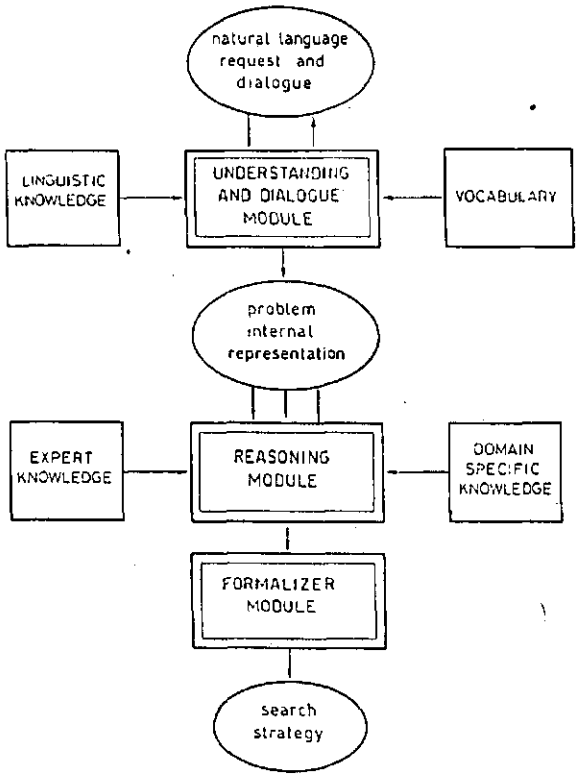


Figure 1 - Overall architecture of IR-NLI.

The world knowledge (mostly of terminologic nature) necessary for carrying out these tasks is contained in a base of domain specific knowledge. The complexity of the tasks of the REASONING MODULE does not allow to consider simply a standard expert system architecture for its implementation. In fact, the problem has a highly multifaceted nature, and the system must be able to switch control among many areas of competence (corresponding to several approaches for strategy design, to different actions to be performed on the PIR, and to distinct phases structuring the interaction with the user) at different levels of abstraction (from high level decisions, concerned with the overall organization of system operations to low level choices related to the execution of actions for the expansion of the PIR). These issues lead to the exigency of providing the rule-based system with an explicit method for representing and using meta-knowledge. The approach to meta-knowledge employed in IR-NLI is based on a novel mechanism, called task, which is supported by a partition of the expert knowledge base. This contains IF-THEN rules and is partitioned into classes which correspond to specific competence areas and contain rules used to solve a particular problem in different contexts. Rules may specify either modifications of the PIR (actions) or modifications of the flow of control (directives). A task is generated run-time by associating a class, with a termination predicate and a list of parameters. A task can be created by a specific directive ("activate") present in the THEN part of some rules. Task execution consists in applying the rules of the class to parts of the PIR specified in the list of parameters. The task ceases to exist when the termination predicate becomes true, or when a termination directive ("terminate") is present in the rule currently being executed.

The mode of operation of the control mechanism is the usual recognize-act loop, under a data-driven regime. The conflict resolution phase is carried out by using a subset of the class, containing conflict rules. The control mechanism may execute rules which contain the activation of other tasks, thus generating a tree-like structure of task activations. Control is resumed when the subtask finishes.

A datailed illustration of the task mechanism and a list of the most significant tasks employed in IR-NLI can be found in [1].

3. An Analysis of the Tasks Mechanism

Let us now analyze the control mechanism above described focusing both on its advantages and limitations. The discussion will lead to the identification of some requirements for a new control mechanism.

The main features of the task mechanism include:

a) locality: tasks allow use of knowledge (both domain and meta) in specific environments, triggering a particular set of rules (the class).

b) sharpness: the task activation method (the directive "activate") allows a sharply focused knowledge retrieval, greatly decreasing the degree of non-determinism present in the system operation. This directly influences the performance of the system.

c) uniformity: both domain knowledge and meta-knowledge are expressed in the same language. This results in an easy definition of conflict resolution strategies.

Two main implications follow:

i) The class/rule organization of knowledge yields a two-level search of the relevant knowledge: the search of the class, and within that, the search of a rule to fire. More precisely, the first is not a search, being the task activated by name.

ii) The "activate" directive (or "terminate") expresses meta-knowledge since it indicates which knowledge class is going to be used (or abandoned). Furthermore the sequencing of actions and directives which may appear in the THEN part of a rule is also meta-knowledge. Thus meta-knowledge is not confined within conflict rules, but rather it is mixed with domain knowledge and may take the form of procedural knowledge.

Tasks have shown the following shortcomings:

a) reference by name: this means that tasks are used in the same way as subprograms within a procedural language. Although decreasing the non-determinism of the computation, this introduces some other problems:

- lack of incrementability: when adding a new task it is necessary to update the tasks already existing in order to make them knowledgeable of the new one;

- static organization of knowledge: once the programmer has stated that a task T can activate a set of tasks {T1,T2,...,Tn}, by no means it might happen that T activates a task not belonging to the above set. Moreover, it is the programmer's responsibility to make the right association between task content (meaning) and task expected results [2]. The system is not able to reason about the meaning of a task.

b) implicit strategy: the mixture of domain knowledge and meta-knowledge which exists in "activate" prevents the system from reasoning on its goals, enabling it to follow different solution paths only according to a data-driven regime. IR-NLI follows the strategy given it by the programmer; it has indeed the possibility to modify it: the higher the number of nested tasks, the higher is the number of alternatives it can follow, but without exerting explicit control on them. The primary consequences of this are:

- difficulty in stating a complete and correct problem solving solution strategy;

- lack of goal-driven reasoning, that could make the system able to devise its own strategies.

c) failure/success recognition: IR-NLI is not able to recognize a failure (an unescapable situation from which it is not possible to derive useful results) or a success (a situation from which no further work is needed). Tasks have, indeed, an elementary capability of recognizing failures and successes: the termination predicate or the IF part of the rules whose THEN part contains the directive "terminate" account for this. They yield just a "yes/no" answer. The implications of this are:

- no qualitative evaluation of the results is available, which means that the system is not able to reason on the operations so far performed and to understand the causes of failure or success;

- intelligent backtracking is not possible, as the system usually performs a blind search of alternatives, including irrelevant ones;

- there is no account for previous successes or failures, in order to affect the current subproblem solution.

d) stack-based control: the activation of tasks implies a stack-based control structure. This limitation is obvious: in every moment we can affect only the task on top of the stack, being prevented the access to the other elements. This yields:

- no possibility to perform suspensions or resumptions of tasks;

- no possibility to perform any scheduling, or rescheduling, among tasks;

- no possibility to have a set of competitive and cooperative processes, running concurrently.

The points just introduced lead to the identification of some requirement of a new system architecture. Four major points can be identified, closely related to the limitations listed above.

Reference mechanism: a more sophisticated reference mechanism should be used. The major alternatives are reference by description and reference by content [2,3]. In the first case each knowledge unit (in our case tasks) comes provided with a list of attributes which describes the unit (e.g., post conditions, resources used, etc.). In the second case the system is able to derive the properties of a knowledge unit by looking at its parts, using a 'reasoning about content' capability.

Strategy representation: the strategy representation plays a major role when the system has to reason about it. Since the strategy is often domain dependent, the knowledge engineer should supply it to the system. It is also important to separate control knowledge (the strategy) from domain knowledge and to provide an explicit representation for it.

Result evaluation: based on the recognition of failures and successes recognition the system should perform a qualitative evaluation of (partial) results, in order to intelligently backtrack when it is required to, and to consider, in the current subproblem solution, previous failures or successes.

Control mechanism: in order to make the system able to reason about its strategy, to use previous results, to perform backtracking, to consider the organization of knowledge and the content of rules it is necessary to address the control issue. Recently it has been pointed out that a distributed view of the problem solving activity has several advantages [6,7,4]. Moreover, the following points should be considered: identification of suitable communication protocols, identification of the information to be exchanged, and cooperation models utilized.

4. Toward a Functionally Distributed Architecture

On the basis of the analysis developed in the previous section, we examine here the issue of effective and natural modelling of the intermediary's behaviour from the point of view of distributed problem solving. In fact, the task mechanism, initially viewed as an effective way for representing meta-knowledge, turns out to be centered on the basic concepts of decomposition and distribution: knowledge of the problem domain is split into a collection of separate competence areas, and a system for having them to cooperate in the problem solving activity has been implemented. However, the issue of distribution is not explicitly faced, and thus the possible advantages of a distributed approach are not fully exploited.
An analysis of the major features of the application domain of online information retrieval from the point of view of distribution yields to the following considerations:

- decomposition of the domain knowledge into competence areas (C-A) is motivated only by logical reasons, as the problem is per se not physically distributed;

- decomposition is, therefore, not unique as several criteria may be adopted for identifying the single C-As;

- two main criteria can be considered for logical decomposition:
  * functional decomposition, aimed at improving skill and naturalness of system behaviour;
  * performance-oriented decomposition, mostly aimed at improving system efficiency;
- functional decomposition has to be preferred over performance-oriented decomposition, (at least in the first stage of the project), as the focus of attention is more on the adequacy and effectiveness of cognitive modelling rather than on the efficiency of implementation;
- C-As that result from functional decomposition do not constitute a flat collection of independent specialists, but:
  * they cover a wide range of different abstraction levels (consider, for example tasks such as "approach selection" and "generalize");
  * they can be partially ordered, thus yielding a hierarchical organization, according to a relation <is-a-subtask-of>;
- C-As are not homogeneous from the point of view of their individual capabilities: they operate on certain and complete input data but the knowledge they utilize does not always guarantee that the outputs they produce are correct and complete (consider, for example, "respell" that performs a well known completely accurate activity, and "Citation-Pearl-Growing" that generally operates with incomplete judgemental knowledge);
- C-As are not homogeneous from the point of view of the mutual knowledge they have about each other: each C-A directly knows only a subset of the existing C-As and thus has not a complete view on the global capabilities of the system;
- as a consequence of the previous point, the communication and control mechanisms among C-As may conform to several different strategies, including direct reference by name, content reference, goal-directed addressing, request-driven activation, etc.;
- parallel activation of C-As is possible, both in the sense of parallel exploration of alternative solution paths and in the sense of parallel execution of independent subtasks.

The above features denote a kind of functionally distributed system that can not be easily modelled in terms of known paradigms. Among the several approaches to distributed problem solving proposed in the literature the following ones seem to be closely related to our investigation: FA/C [7], ETHER [6], and contract net [4]. No one of them is however appropriate to cover the specific features of our application. To this purpose we propose a novel approach to distributed system modelling that is based on three major points:

1. each node operates on certain, correct, and complete input data;
2. the processing performed by a node can yield uncertain results, as it utilizes possibly incomplete or uncertain knowledge;
3. the behaviour of the network is not strictly cooperative as nodes are not expected to communicate partial tentative results and to refine them through an iterative coroutine-type interaction; nevetherless, some degree of cooperation among nodes exists as the behaviour of a node can be directly influenced by the results obtained by other nodes.

These features make our approach different from both CA/NA and FA/C and from other possible combinations of these approaches [7]. Thus we denote our distributed mechanism "complete but unaccurate and weakly cooperative".

5. Conclusion

The proposal for a new paradigm of distributed problem solving sketched in the previous section needs to be developed and refined along several directions before experimental activity can be started. Among the presently ongoing research efforts we mention:

- definition of a suitable communication protocol among nodes [8,4];
- design of the appropriate communication/control mechanism capable of making the nodes interact and cooperate according to the proposed paradigm.

The adequacy of the proposed approach will be then tested on a new version of IR-NLI.

References

1. G. Brajnik, G. Guida, and C. Tasso, "An expert interface for effective man-machine interaction," in Cooperative Interactive Systems, ed. L. Bolc, Springer Verlag, Berlin, FRG. To appear.
2. R. Davis, "Content reference: reasoning about rules," Artificial Intelligence, no. 15, pp. 223-239, 1980.
3. R. Davis, "Meta-rules: reasoning about control," Artificial Intelligence, no. 15, pp. 179-222, 1980.
4. R. Davis and R. G. Smith, "Negotiation as a Metaphor for Distributed Problem Solving," Artificial Intelligence, no. 20, pp. 63-109, 1983.
5. G. Guida and C. Tasso, "An expert intermediary system for interactive document retrieval," Automatica, vol. 19, no. 6, pp. 759-766, 1983.
6. W. A. Kornfeld and C. E. Hewitt, "The Scientific Community Metaphor," IEEE Trans. on Systems, Man, and Cybernetics, vol. 11, no. 1, pp. 24-34, 1981.
7. V. R. Lesser and D. D. Corkill, "Functionally accurate, cooperative distributed systems," IEEE Trans. on Systems, Man, and Cybernetics, vol. 11, no. 1, pp. 81-95, 1981.
8. R. G. Smith and R. Davis, "Frameworks for cooperation in a distributed problem solver," IEEE Trans. on Systems, Man, and Cybernetics, vol. 11, no. 1, pp. 61-70, 1981.

(N E) N O Č P R O L O G A

MARKO GRODELNIK
INŠTITUT JOŽEF STEFAN - LJUBLJANA
JUGOSLAVIJA

UDK: 681.3.06

POVZETEK

O prologu, kot enem najpomembnejših jezikov umetne inteligence je v zadnjih letih precej govora. Veliko strokovnjakov pa je v dvomih o uporabnosti prologa v profesionalnem programiranju. Namen referata je osvetliti dobre in slabe lastnosti prologa in prikazati njegovo uporabnost v vsakdanji programerski praksi.

ABSTRACT

During recent years, the prolog language has been much published and spoken about. But many computer experts have doubts about professional usefulness of this language. The purpose of this manuscript is to make clear good and bad properties of prolog and to show its usefulness in everyday programmer practise.

1. UVOD

V zadnjem času je v različnih računalniških publikacijah precej govora o umetni inteligenci, peti računalniški generaciji in s tem v zvezi s prologom kot enim najpomembnejših jezikov na tem področju računalništva. Tudi na najbolj razširjenih računalnikih vseh velikosti so se v zadnjih letih pojavile bolj ali manj uspešne implementacije tega jezika. Med računalniškimi strokovnjaki pa se v veliki meri pojavlja dvom o smiselnosti uporabe prologa v profesionalnem programiranju. Ta dvom je predvsem posledica dveh napačnih pogledov na prolog:

1. Večina programerjev zavrača prolog samo zaradi njegove relativne neučinkovitosti na današnjih računalnikih. Žal je tako ocenjevanje jezika zelo enostransko.
2. Tisti, ki poznajo prolog samo iz knjig ali kratkih tečajev, so navadno razočarani nad tem, ker od obljubljene nepostopkovnosti prologa v praksi ne ostane dosti. Vsekakor taki ljudje pričakujejo od prologa preveč.

Jasno je, da ima tudi prolog slabosti. Vendar nam za ceno teh slabosti nudi nov pristop k obravnavi problema, ki se izkaže kot zelo učinkovit. Namen tega referata je prikazati prolog z več strani, prikazati uporabo prologa v praksi in opozoriti na dobre in slabe lastnosti, ki se pri tem pojavljajo.

2. PROLOG

Prva implementacija prologa je bila narejena leta 1972 na univerzi v Marseillu /6/ kot posledica ideje, da bi matematično logiko razvili v programski jezik. Prvi implementaciji so sledile nove. Do leta 1979 jih je bilo že okrog 20.

V tem času so se razvile tudi nove tehnike in optimizacijski mehanizmi za implementiranje prologa. Največji doprinos k temu je nedvomno dala implementacija na računalniku DEC-10 /C/, ki vključuje prolog interpreterja tudi do tistega časa prvi prevajalnik za prolog. DEC-10 prolog je postal tudi nekakšen neuraden standard za prolog. V osemdesetih letih so se nove implementacije kar vrstile. Najpomembnejša je Quintus prolog, ki je izboljšava prologa na računalniku DEC-10. Prolog je bil tudi izbran kot osnovni jezik pete računalniške generacije na Japonskem. KLO (Kernel Language zero), ki je strojni jezik prvega računalnika pete generacije (Delta machine) je namreč razširjen prolog /5/.

Matematična osnova prologu je predikatni račun prvega reda /3/. Prolog uporablja podjezik predikatnega računa t.i. Hornove stavke, ki izražajo pogojne trditve tipa :

P če P1 in P2 in P3 in ... in Pn

V primeru, ko je n=0, so to brezpogojne trditve oz. dejstva.

Program v prologu tako lahko razumemo kot niz definicij, ki si jih lahko razlagamo povsem deklarativno. Ob programiranju teh definicij pa se moramo zavedati tudi postopkovnega pomena takega programa. V nasprotnem primeru bi nas prolog pripeljal v nekatere neljube situacije, iz katerih se brez znanja o postopkovnem pomenu programa ne bi mogli rešiti.

Pomembni lastnosti prologa sta tudi nedeterminizem, ki je realiziran s pomočjo avtomatskega vračanja (automatic backtracking) in manipuliranje s seznami, ki je podobno kot v LISP-u.

Vhodno-izhodne funkcije in še nekatere druge, ki jih ne bi mogli izvesti v čistem prologu, so realizirane s pomočjo vgrajenih podprogramov.

3. SLABE STRANI PROLOGA

Nedvomno je največja pomanjkljivost prologa v njegovi relativni počasnosti. Ta slabost je nasploh značilnost takih jezikov, od katerih pa jo prolog še najbolje rešuje. Daleč največ časa pri izvajanju porabi operacija prilagajanja (unification) dveh podatkovnih struktur, ki porabi preko 90% časa. Večji del preostanka porabi administracija nad podatki (prolog ima 4 podatkovna območja). Počasnost so poskušali omiliti z uvajanjem optimizacijskih mehanizmov /6/ in pri tem dosegli dokaj dobre rezultate. Kljub vsemu pa hitrosti postopkovnih jezikov prolog na današnjih računalnikihverjetno ne bo nikdar dosegel.

Druga pomanjkljivost je velika poraba pomnilniškega prostora. Z uvajanjem nove tehnologije in večanjem pomnilniških kapacitet ta slabost izgublja na pomenu. V primerih, ko računalnik nima dovolj velikega pomnilniškega prostora, pa je ta slabost celo bolj kritična kot hitrost izvajanja. Razlog za veliko porabo pomnilnika je v tem, ker mora prolog ohranjati podatke zaradi avtomatskega vračanja. Optimizacijski mehanizmi /6/ sicer poskrbijo, da se vsi nerabni podatki odstranijo, vendar poraba ostaja še vedno velika.

Prolog se ne izkaže pri obsežnem delu s števili in pri obsežnih vhodno-izhodnih operacijah. V takih primerih program izgleda podobno kot v klasičnih postopkovnih jezikih.

Nekatere starejše implementacije imajo to pomanjkljivost, da ne nudijo povezave z operacijskim sistemom. V novejših implementacijah je to v polni meri rešeno.

Predvsem v Pascalu in podobnih jezikih vzgojeni programerji prologu očitajo preveliko splošnost in svobodo pri delu s podatki. Navadno pa se ta lastnost v praksi izkaže bolj kot prednost in ne kot slabost.

Tisti, ki pričakujejo od prologa preveč, bi radi pisali kakršnekoli programe v prologu brez upoštevanja postopka izvajanja. V 2. razdelku smo omenili , da se brez upoštevanja postopka da brati programe, ne pa tudi pisati. Prologov sistem more pri izvajanju izvršiti določen postopek, zato so take pričakovanja nesmiselna.

4. DOBRE STRANI PROLOGA

Moč prologa se nedvomno najbolj pokaže pri manipuliranju s podatki. Simbolično procesiranje, ki je podkrepljeno s strukturiranjem podatkov v poljubne drevesne in grafne strukture omogoča presenetljivo enostavno delo z velikimi količinami podatkov. Pri tem se predvsem izkaže delo s t.i. seznami, ki nimajo vnaprej določene dolžine in lahko vsebujejo poljubne količine podatkov. Ob tem, da niso potrebne še nikakršne deklaracije podatkov, so taki programi navadno nekajkrat krajši in precej bolj čitljivi kot za enak problem v klasičnih postopkovnih programskih jezikih.

Sintakso prologa je zelo enostavna. Opisana je približno v petnajstih produkcijskih pravilih (PNF).

Ena izmed prednosti prologa je tudi avtomatsko vračanje. S to lastnostjo prolog sam poišče vse rešitve v drevesu možnosti. Naloga programerja je le, da omeji preiskovanje drevesa, če je to potrebno. Avtomatsko vračanje močno olajša programiranje algoritmov kombinatorične narave.

Za podprograme v prologu je značilna velika splošnost. Isti podprogram lahko uspešno opravlja naloge za vhodne podatke povsem različnih tipov in količin. Ta splošnost se predvsem odraža v dolžini programa.

Kareva prologovega programa programerja dobesedno sili v tehniko programiranje "od zgoraj navzdol" (top-down) /1/. Struktura prologovih stavkov

P če P1 in P2 in P3 ... in Pn

deli problem P na n manjših podproblemov. Ta delitev se izvaja dokler ne naletimo na prologova dejstva ali na vgrajene podprograme. Zaradi take strukture je tudi prevajanje programov iz prologa v strukturirane jezike (npr. Pascal) dokaj enostavno. Težave nastopijo le v primeru, ko prevajamo programske dele, ki imajo nedeterministični značaj in pri prevajanju splošnih podprogramov. Sicer pa prologovi programi še vedno ohranjajo deterministični značaj, če programer oz. problem to zahteva.

Vsak prologov sistem vsebuje tudi podsistem za odkrivanje napak (debugger), s katerim lahko zelo hitro in zanesljivo odkrijemo napake v programu. V splošnem poznamo dva tipa sistemov za odkrivanje napak : postopkovni in deklarativni. Pri prvem /3/ zasledujemo postopek izvajanja pravil. Pri drugem /4/ pa nastavimo pravilne podatke, s katerimi poskuša sistem sam ugotoviti, kje je prišlo do napake in nam celo svetuje kako naj jo odpravimo.

Zelo koristna lastnost so lahko tudi gramatična pravila. To je formalizem v okviru prologove sintakse, ki omogoča opisovanje poljubnih gramatik. Pregledovanje jezika se izvaja nedeterministično. Uporaben pa je predvsem v primeru, ko programiramo pregledovalnike za določen jezik. Formalizem je precej močnejši kot PNF (ali BNF). /3/

Prolog lahko direktno uporabimo za programiranje relacijskih baz podatkov /2/. To je v prologu dokaj enostavno, kajti tako kot prolog ima tudi relacijski model baz podatkov za osnovo matematično logiko.

5. UPORABA PROLOGA V PRAKSI

Prolog je v praksi najbolj uporaben kot jezik za prototipno programiranje oz. za specifikacijo sistemov /1/. Relativna počasnost in velika poraba pomnilniškega prostora sta glavni oviri za to, da bi bil prolog tudi jezik za končno delujoče aplikacije.

Relativno kratka in čitljiva koda ter splošnost prologovih programov omogočajo prototipno programiranje. Za tak način programiranja je predvsem zahtevan hiter odziv programerja na spremembe specifikacij.

V vsakdanji programerski praksi se velikokrat srečamo s pomanjkljivimi specifikacijami. Navadno pa se to pokaže šele, ko je večji del sistema že izdelan. Prolog lahko v takem primeru uporabimo za specifikacijo sistema in za zelo majhno ceno se pokažejo pomanjkljivosti, ki jih lahko takoj odpravimo. /1/

Opisane lastnosti prologa so še posebno uporabne pri programiranju ekspertnih sistemov /2/, kjer sta največja problema zbiranje specifikacij (t.i. Feigenbaumovo ozko grlo) in iskanje algoritma za učinkovito delovanje sistema. V obeh primerih se prolog izkaže kot zelo koristno

orodje za razvoj sistema.

6. ZAKLJUČEK

Iz prej povedanega sledi, da je prolog kljub slabostim, ki so splošno značilne za inteligentnejše jezike uporaben tudi v vsakdanji programerski praksi. Manj primeren oz. neprimeren je za pisanje končno delujočih implementacij sistemov. Vnekakor pa prolog pokaže svoje dobre lastnosti pri definiciji in razvijanju sistemov katerekoli vrste. Dobra lastnost, ki mu to omogoča je relativno kratka, čitljiva in splošno uporabna koda.

ZAHVALA

Posebno zahvalo izražam sodelavcem skupine za umetno inteligenco na Inštitutu Jožef Stefan za pomoč pri zbiranju informacij in za kreativne pripombe.

LITERATURA

1. Bojadžijev D.,Lavrač N., Kozetič I.
   IZKUŠNJA S PROLOGOM KOT JEZIKOM ZA SPECIFIKACIJO INFORMACIJSKIH SISTEMOV
   IJS - delovno poročilo 2509, 1981
2. Bratko I.
   INTELIGENTNI INFORMACIJSKI SISTEMI
   Fak. za elektrotehniko, Ljubljana 1983
3. Clocksin W.F., Mellish C.S.
   PROGRAMMING IN PROLOG
   Springer - Verlag, Berlin Heidelberg New York 1981
4. Kononenko I.
   STRUKTURNO AVTOMATSKO UČENJE
   Fak. za elektrotehniko, Ljubljana1985
5. Manuel T.
   CAUTIOUSLY OPTIMISTIC TONE SET FOR 5th GENERATION
   Electronics Week, December 3, 1984
6. Warren D.H.D.
   IMPLEMENTING PROLOG
   D.A.I. research report no. 39, Edinburgh 1977

PREPOZNAVANJE GLASOVA SEKVENCIJSKOM ANALIZOM AUTOKORELACIJSKIH VEKTORA OGRANIČENOG BROJA UZASTOPNIH SEKCIJA

I. Marić

Institut "Ruđer Bošković", Zagreb

UDK: 681.326.7

U ovom radu prikazan je sustav za prepoznavanje nekih glasova hrvatskog jezika usporedivanjem s referentnim uzorcima. Korišten je standardni model za prepoznavanje govora koji uključuje određivanje vektora karakterističnih svojstava, kreiranje referentnih uzoraka, usporedivanje test i referentnih uzoraka, te konačnu odluku o prepoznatom uzorku. Primjenjena je modificirana verzija jednorazinskog algoritma za dinamičko programiranje. Ispitivanja su vršena na ograničenom broju glasova. Prikazani su i rezultati mjerenja.

PHONE RECOGNITION USING SEQUENTIAL AUTOCORRELATION VECTOR ANALYSIS OVER LIMITED NUMBER OF CONSECUTIVE FRAMES: In this work the speaker dependent phone recognition system is presented. The system is based on the phone recognition by means of the reference templates. The canonic pattern-recognition model for speech recognition is used, including generation of the characteristic feature vectors, reference template creation, pattern similarity determination, and the final decision about the recognized set of phones. A modified version of the one stage dynamic programming algorithm is applied. The evaluation of the algorithm is performed over the restricted set of phones. The results of the measurements are also presented.

UVOD

Iako je nagli razvoj tehnologije integriranih krugova, kao i teorije digitalne obrade signala doveo do ubrzanog razvoja različitih metoda za prepoznavanje govora, problem je još uvijek daleko od konačnog rješenja. Postupci su složeni, zahtijevaju obradu velikih količina podataka u kratkom periodu kao i ogromnu memoriju. Unatoč tome postoji već čitav niz komercijalnih rješenja za prepoznavanje ograničenog broja riječi određenog govornika s pouzdanošću većom od 95%. U novije vrijeme razvijeni su i sistemi, nezavisni o govorniku, za prepoznavanje povezanih riječi.

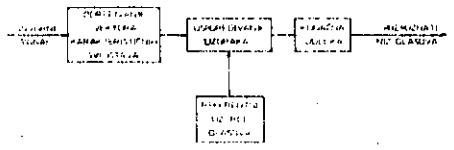
Primjena dinamičkog programiranja u prepoznavanju govora [1-3] predstavlja trenutno najperspektivnije rješenje a istraživanja su orjentirana u smjeru pojednostavnjenja algoritama dinamičkog programiranja i povećanja njihove efikasnosti. problem se sastoji u pronalaženju referentnog uzorka ili niza referentnih uzoraka govora, pohranjenih u memoriji računala, koji najbolje odgovaraju test uzorku govora. Referentni uzorci predstavljaju niz višedimenzionalnih vektora, pri čemu svaki vektor karakterizira jedan trenutak ili kratki vremenski isječak govora. Ti vektori predstavljaju vektore karakterističnih svojstava govornog signala koji mogu biti generirani analizom u frekvencijskoj (engl. Filter Bank-Analysis of Speech) ili u vremenskoj domeni (engl. LPC Analysis of Speech) [1]. Dinamičkim programiranjem traži se optimalni vremenski put kroz referentne uzorke po kriteriju minimalne akumulirane razlike između vektora karakterističnih svojstava referentnih i test uzoraka na tom putu. Pri tome se postavljaju ograničenja na put čime se reducira broj mogućih rješenja.

U ovom radu bit će prikazana realizacija jednog sustava za prepoznavanje glasova hrvatskog jezika. S obzirom da u hrvatskom jeziku svakom slovu odgovara određeni glas i obrnuto, te da glasovi unutar riječi zadržavaju neka svoja bitna obilježja na osnovi kojih mogu biti prepoznati, analiza se može provesti dinamički po vremenskim isječcima govora prepoznavajući glasove tj. slova u nizu. Ta primitivna analiza može se shvatiti kao neka vrsta "elektroničkog uha" koji rezultate svoje obrade može slati na daljnju analizu nekom centralnom sustavu u kojem su pohranjena fonetska svojstva jezika.

MODEL ZA PREPOZNAVANJE GLASOVA

Model za prepoznavanje glasova (slika 1), koji je ovdje korišten, po svojim bitnim obilježjima

se ne razlikuje od standardnog modela za prepoznavanje govora [1]. Kao što se iz sl.1 vidi,



Slika 1. Model za prepoznavanje glasova

sustav se sastoji iz nekoliko zasebnih logičkih cjelina:

- odredivanje vektora karakterističnih svojstava govornog signala, bez obzira da li se radi o kreiranju referentnih uzoraka ili obradi test uzorka
- generiranje referentnih uzoraka glasova, provodi se u modu za učenje (referentni uzorci predstavljaju reprezentativne uzorke za svaki glas posebno)
- usporedivanje uzoraka vrši se u test modu tj. vektori karakterističnih svojstava test uzorka uspoređuju se s referentnim
- konačna odluka o prepoznatim glasovima vrši se u posljednjoj fazi obrade, pri čemu u odlučivanju mogu biti uključena i pravila izgovora

U ovoj fazi sustav je realiziran programski s minimumom neophodnih sklopova.

ODREDIVANJE VEKTORA KARAKTERISTIČNIH SVOJSTAVA

Generiranje vektora karakterističnih svojstava bilo test bilo referentnih uzoraka, bazirano je na poznatoj LPC metodi tj. linearnom kodiranju s predvidanjem unaprijed (engl. Linear-Predictive Coding) [6], odnosno na činjenici da se uzorak govora u svakom trenutku može aproksimirati linearnom kombinacijom prethodnih uzoraka tj.,

$$\tilde{x}(n) = \sum_{k=1}^{p} a_k \cdot x(n-k) \qquad (1)$$

gdje je x(n) stvarni uzorak (nekvantizirani), $\tilde{x}(n)$ predvideni uzorak, a_k koeficijenti aproksimacije a p red aproksimacije. Pogreška predvidanja definirana je na slijedeći način

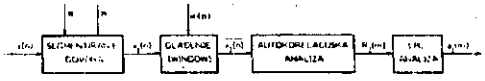
$$e(n) = x(n) - \tilde{x}(n) = x(n) - \sum_{k=1}^{p} a_k \cdot x(n-k) \qquad (2)$$

Odredivanje koeficijenata a_k bazira se na minimizaciji srednje kvadratne pogreške zbog predvidanja na kratkom segmentu govornog signala tj. traže se vrijednosti a_k koje minimiziraju jednakost:

$$e_1 = \sum_{n=0}^{N-1} (x_1(n) - \tilde{x}_1(n))^2 =$$

$$= \sum_{n=0}^{N-1} (x_1(n) - \sum_{k=1}^{p} a_k \cdot x_1(n-k))^2, \qquad (3)$$

gdje $x_1(n)$ predstavlja 1-ti segment govornog signala. Na slici 2 ilustriran je postupak odredivanja LPC koeficijenata.



Slika 2. Postupak odredivanja LPC koeficijenata

Govorni signal se najprije segmentira u sekcije duljine N uzoraka s preklapanjem uzastopnih sekcija u N-M uzoraka tj., slijedeća sekcija pomaknuta je za M uzoraka u odnosu na prethodnu. Ako je M>N ne postoji preklapanje izmedu sekcija. Preklapanje sekcija nužno ima za posljedicu gladenje vektora karakterističnih svojstava. Da bi se umanjili rubni efekti svaka sekcija govora množi se s težinskom funkcijom w(n) (engl. window) tj. $\bar{x}_1(n)=x_1(n)\cdot w(n)$. Težinska funkcija w(n) je periodička funkcija s periodom N. Tipična težinska funkcija je Hamming window definirana na slijedeći način:

$$w(n) = 0{,}54 - 0{,}64 \cdot \cos\left[\frac{2\pi n}{N-1}\right] \qquad (4)$$

U slijedećem koraku vrši se autokorelacijska analiza sekcije govora pomnožene s težinskom funkcijom tj., odreduju se koeficijenti

$$R_1(m) = \sum_{n=0}^{N-1-m} \bar{x}_1(n) \cdot \bar{x}_1(n+m), \quad m=0,1,\ldots,p \qquad (5)$$

gdje p označava red analize sustava ($p \leqslant 12$). Skup koeficijenata

$$X_1 = \left\{R_1(0), R_1(1), \ldots, R_1(p)\right\} \qquad (6)$$

se često koristi kao rezultat LPC analize jer se karakteristični vektori svojstava i test i referentnog uzorka mogu iz njih odrediti. Ako se autokorelacijski koeficijenti normaliziraju tako da $R_1(0)$ postane jednak jedinici može se definirati vektor karakterističnih svojstava 1-te sekcije na slijedeći način:

$$V_1 = \left\{r_1(0), r_1(1), \ldots, r_1(p)\right\} \qquad (7)$$

gdje je $r_1(0)=1/R_1(0)$ (normalizacijski faktor) a $r_1(i)=R_1(i)/R_1(0)$, $i=1,2,\ldots,p$. Normalizacija autokorelacijskih koeficijenata umanjuje utjecaj varijacije intenziteta govornog signala na prepoznavanje.

GENERIRANJE REFERENTNIH UZORAKA

Niz vektora karakterističnih svojstava određenog glasa čini referentni uzorak glasa. Prilikom generiranja referentnih uzoraka kao i pri analizi test uzoraka važno je locirati početak i kraj govornog signala. U tu svrhu mjeri se energija signala u svakom vremenskom isječku prema izrazu

$$E_1 = \sum_{n=0}^{N-1} \bar{x}_1(n)^2 = R_1(0) \qquad (8)$$

U periodu kada je energija E_1 veća od unaprijed zadane minimalne energije (šum okoline) definiran je valjan signal.

Referentni uzorak određenog glasa može se zapisati na slijedeći način:

$$U(k) = \left\{ V(k,j) \right\}, \; j=1,\ldots,J(k); \; k=1,\ldots,K \qquad (9)$$

pri čemu V(k,j) označava vektor karakterističnih svojstava j-te sekcije k-tog glasa. Ukupni broj sekcija J(k) zavisan je o duljini uzorka. Svaki glas ima svoj alfabetski znak tj. slovo:

$$L(k) \triangleq U(k) \qquad (10)$$

Vokali mogu imati nekoliko odgovarajućih uzoraka zavisno npr. o vrsti naglaska.

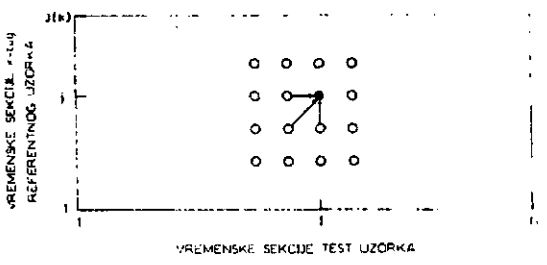
Test uzorak govora može se prikazati na sličan način,

$$T = \left\{ V(i) \right\}, \; i=1,\ldots,I \qquad (11)$$

gdje je V(i) odgovarajući vektor i-te sekcije test uzorka a I ukupan broj vektora test uzorka.

USPOREDIVANJE UZORAKA

Usporedivanje test i referentnih uzoraka vrši se modificiranom verzijom jednorazinskog algoritma za dinamičko programiranje [2] , po kriteriju minimalne akumulirane razlike između test i referentnog uzorka. Određivanje se vrši na fiksnom broju točaka od početka referentnog uzorka. Ukoliko je minimalna akumulirana razlika manja od unaprijed definirane vrijednosti prepoznat je početak odgovarajućeg glasa. Prepoznati glas traje do prepoznavanja početka novog glasa, ukoliko se ne radi o uzastopnom prepoznavanju početka jednog te istog glasa. Istodobno se registriraju i pauze između glasova te se na osnovi informacija o glasu, intonaciji i pauzama mogu prepoznati pojedine riječi ili niz riječi. Na slici 3 ilustrirana su ograničenja na pomake između točaka prilikom traženja optimalnih putova unutar k-tog referentnog uzorka. Iz slike 3 vidi se da se u točku (i,j,k) može doći samo iz točaka (i-1,j,k), (i-1,j-1,k) i (i,j-1,k). Ako s D(i,j,k) označimo minimalanu akumuliranu pogrešku do točke (i,j,k) po bilo kojem putu a s d(i,j,k) razliku između i-tog

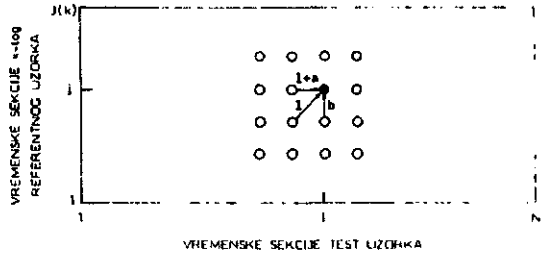


Slika 3. Ograničenja na pomake između točaka unutar uzorka

vektora test uzorka i j-tog vektora referentnog uzorka tada, uvažavajući pravila definirana na slici 3, vrijedi slijedeća relacija:

$$D(i,j,k) = d(i,j,k) + \min\left\{ D(i-1,j,k), \; D(i-1,j-1,k), D(i,j-1,k) \right\} \qquad (12)$$

Optimalni put vrlo rijetko odstupa od dijagonalnog tj. ima nagib 1, budući da su promjene frekvencije govora općenito malene. Da bi se na određeni način favorizirali putovi čiji je nagib približno jednak jedinici uvode se težinski faktori s kojima se množe lokalne razlike prije primjene rekurzivne formule dinamičkog programiranja (12). Ne postoji općenito rješenje ovog problema. Na slici 4 ilustrirano je jedno moguće rješenje.



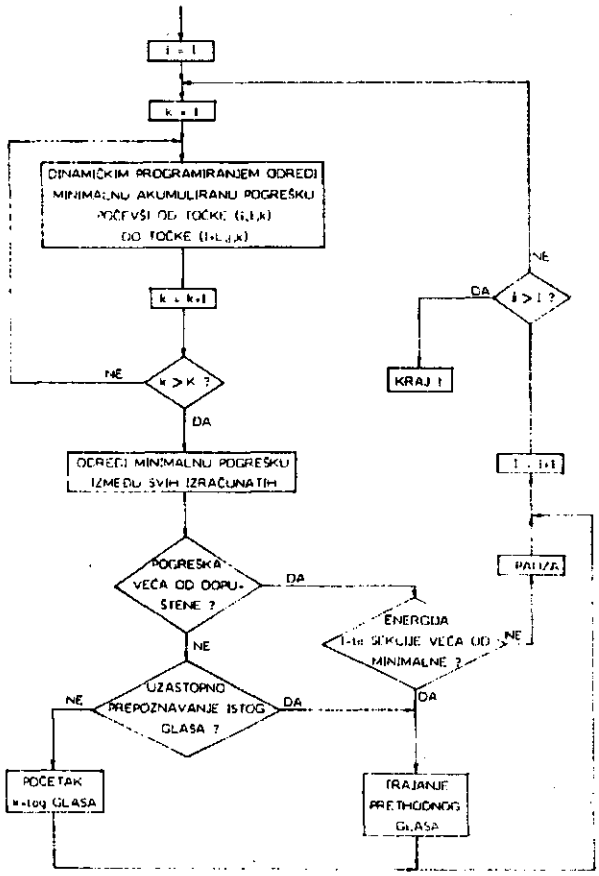
Slika 4. Težinski faktori zbog vremenskog izobličenja

Minimalna akumulirana razlika do točke (i,j,k) u ovom slučaju definirana je rekurzivnom relacijom:

$$D(i,j,k) = \min\left\{ \begin{array}{l} (1+a)\cdot d(i,j,k)+D(i-1,j,k), \\ d(i,j,k)+D(i-1,j-1,k), \\ b\cdot d(i,j,k)+D(i,j-1,k) \end{array} \right\} \qquad (13)$$

REALIZACIJA ALGORITMA

Konačni cilj je pomoću algoritma prepoznati niz povezanih glasova. Da bi se to postiglo, minimalna akumulirana pogreška računa se od početka test uzorka, prema dijagramu toka na slici 5. Prije primjene algoritma određuju se autokorelacijski koeficijenti i generiraju vektori kakarkterističnih svojstava test uzorka. Konačna odluka o prepoznatim glasovima donosi se na osnovi dodatne analize niza podataka koji pred-



Slika 5. Dijagram toka modificiranog jednorazinskog algoritma dinamičkog programiranja za prepoznavanje povezanih glasova

stavljaju rezultate prikazanog algoritma. Cijeli broj L u dijagramu toka označava red analize, u konkretnom slučaju 3.

REZULTATI MJERENJA

Verifikacija uspješnosti algoritma vršena je na svim samoglasnicima (a,e,i,o,u) i nekim suglasnicima (m,n,s,š) hrvatskog jezika. Od navedenih glasova pravljene su suvisle ili nesuvisle kombinacije (riječi) koje su zatim analizirane. U modu za učenje najprije su generirani referentni uzorci za pojedine glasove. Nakon toga testirano je prepoznavanje riječi sastavljenih od spomenutih glasova. Sustav je obučavan za jednog govornika. Analiza se odvijala u produljenom vremenu jer je model realiziran programski. Ulazni signal otipkavan je s frekvencijom 10 kHz i rezolucijom 10 bita nakon što je prethodno filtriran analognim pojasnim propustom s graničnim frekvencijama 80 i 4800 Hz. Provedene su dvije vrste testiranja: prepoznavanje niza gore navedenih glasova u različitim kombinacijama te prepoznavanje riječi sastavljenih od istih glasova a koje pripadaju unaprijed definiranom skupu riječi. U oba slučaja postupak analize je isti osim što je konačna odluka u drugom slučaju ograničena na konačan skup mogućih kombinacija glasova. U tablicama 1 i 2 prikazani su odgovarajući rezultati mjerenja u obje varijante.

GLAS	a	e	i	o	u	m	n	s	š
POSTOTAK PREPOZNAVANJA	95	83	86	97	93	68	64	75	73

Tablica 1. Postotak prepoznavanja glasova u slobodnim kombinacijama

GLAS	a	e	i	o	u	m	n	s	š
POSTOTAK PREPOZNAVANJA	97	89	91	97	96	77	75	86	82

Tablica 2. Postotak prepoznavanja glasova u ograničenom skupu riječi

Kao što se iz tablica 1 i 2 vidi, postotak prepoznavanja glasova u slobodnim kombinacijama niži je od postotka prepoznavanja odgovarajućih glasova u fiksnim kombinacijama. To se događa iz razloga što se sve kombinacije glasova, koje ne pripadaju definiranom skupu riječi nakon analize, zamjenjuju s najsličnijom rječju iz skupa. Eliminirajući tako nepostojeće kombinacije glasova povećava se postotak prepoznavanja. Prepoznavanje glasova jako je zavisno o poziciji glasa unutar riječi te o susjednim glasovima. Daljnja istraživanja trebala bi uključiti prepoznavanje preostalih suglasnika kao i glasovnih prijelaza. To se može postići na dva načina, poboljšanjem algoritma ili povećanjem broja referentnih uzoraka glasova.

ZAKLJUČAK

U ovom radu prikazana je programska realizacija jednog sustava za prepoznavanje glasova hrvatskog jezika. Korišten je standardni model za prepoznavanje govora. Primjenjena je sekvencijska analiza autokorelacijskih vektora ograničenog broja uzastopnih sekcija, što predstavlja modificiranu verziju jednorazinskog algoritma za dinamičko programiranje. Sustav je baziran na laboratorijskom mikroračunalu. Analiza se odvija u produljenom vremenu.
Ispitivanja su vršena na ograničenom broju glasova. U fazi učenja za svaki glas je generiran jedan ili više referentnih uzoraka koji predstavljaju skup vektora karakterističnih svojstava tj., normaliziranih autokorelacijskih koeficijenata dobivenih analizom vremenskih isječaka glasa (djelomična LPC analiza). Testirano je prepoznavanje različitih kombinacija glasova. Rezultati su pokazali zadovoljavajući pos-

totak prepoznavanja glasova, naročito u slučaju ograničenog skupa riječi. Prepoznavanje glasova pokazalo se jako zavisnim o poziciji glasa unutar riječi. Problem bi se mogao izbjeći poboljšanjem algoritma ili povećanjem broja referentnih uzoraka glasova tj., uzoraka koji bi predstavljali glasovne prijelaze. Neriješen problem je i optimalan broj uzastopnih sekcija koje treba analizirati zbog različitih trajanja glasova.

REFERENCIJE:

1. L.R.Rabiner: Tutorial on Isolated and Connected Word Recognition, EURASIP, 1983., pp. 399-406.
2. H.Ney: the Use of a One-Stage Dynamic Programming Algorithm for Connected Word Recognition, IEEE Trans. on ASSP, Vol.32, No.2, pp. 263-271, April 1984.
3. C.Myers, L.R. Rabiner, A.E. Rosenberg: Performance Tradeoeffs in Dynamic Tima Warping Algorithms for Isolated Word Recognition, IEEE Trans. on ASSP, Vol.28, No.6, pp. 623--635, Dec. 1980.
4. P.Regel: A Module for Acoustic-Phonetic Transcription of Fluently Spoken German Speech, IEEE Trans. on ASSP, Vol.30, No.3, pp. 440-450, June 1982.
5. R.W. Schafer, L.R.Rabiner: Digital Representation of Speech Signals, Proceedings of the IEEE, Vol.63, No.4, pp.662-677, April 1975.
6. J. Makhoul: Linear Prediction: A Tutorial Review, Proceedings of the IEEE, Vol.63, No.4, pp. 561-579, April 1975.

PETA RAČUNALNIŠKA GENERACIJA

FIFTH COMPUTER GENERATION

FUNKCIONALNO PROGRAMIRNI SISTEMI

Borut Robič, Jurij Šilc in Branko Mihovilovič
Institut 'Jožef Stefan', Ljubljana

UDK: 681.3.06

Redukcijske računalniške arhitekture zahtevajo funkcionalno programirne jezike. Funkcionalno programirni sistemi predstavljajo enega izmed možnih načinov funkcionalnega programiranja. Čeprav so šibkejši od lambda računa, pa so predvsem zaradi preglednosti in hierarhične zgradbe primernejši za praktično uporabo.

FUNCTIONAL PROGRAMMING SYSTEMS - Some new kinds of computer architectures require functional programming languages. Besides lambda style there is another functional style of programming, namely functional programming systems. These systems have properties which make them more useful than lambda style although they are not as powerful.

Uvod

Sodobne računalniške arhitekture, ki temeljijo na paralelnem procesiranju, zahtevajo programske jezike brez stranskih učinkov. Primer takega jezika je LISP, ki temelji na lambda računu. V lambda računu so izvor novih funkcij lambda izrazi skupaj s pripadajočimi pravili zamene. Z lambda izrazi lahko definiramo vse izračunljive funkcije s poljubno mnogo argumenti. To moč lambda računa pa je zaradi nepreglednosti njegovih izrazov praktično zelo težko izkoristiti.
Lambda račun je le en način ti. funkcionalnega programiranja. Drug pristop k funkcionalnemu programiranju pa predstavljajo funkcionalno programirni sistemi (v nadaljnem FP sistemi). Ti sistemi so šibkejši od lambda računa, toda mnogo preglednejši. Tudi FP sistemi ne poznajo stranskih učinkov.
Program v FP sistemu (v nadaljnem FP program) je funkcija, ki preslika en objekt v drugega. Objekt je bodisi a) nedefiniran , b) atom ali pa c) zaporedje, sestavljeno iz atomov ali zaporedij. Zaporedje je lahko tudi prazno. FP program je lahko a) osnoven, b) sestavljen na podlagi funkcijske oblike, t.j. pravila, s katerim lahko obstoječe FP programe in objekte sestavimo v nov FP program ali pa c) je definiran s pomočjo definicije.
FP program f deluje nad objektom x in vrne rezultat f:x. Pravimo, da f:x opisuje uporabo (aplikacijo) programa f nad objektom x.
Podobno kot lamda račun ima tudi FP sistem zelo enostavno semantiko.

Primerjava proceduralnega in FP programa

Če zapišemo zaporedje stavkov za izračun skalarnega produkta dveh vektorjev v enem od proceduralnih (von-Neumannovih) jezikov

```
c:=0;
for i:=1 to n do
  c := c + a[i]*b[i] ;
```

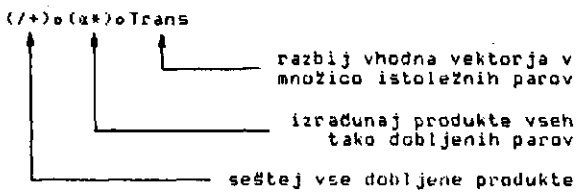
opazimo sledeče lastnosti :

a) računanje temelji na množici različnih pravil, kot so prireditev, indeksiranje , inkrementiranje, testiranje, skoki ;
b) nehierarhično zgradbo, kar pomeni, da kompleksnejši objekti niso sestavljeni iz enostavnejših. Izjema je le desni del prireditvenega stavka c+a[i]*b[i], ki je sestavljen iz dveh enostavnejših c in a[i]*b[i], pri čemer je a[i]*b[i] zopet sestavljen iz dveh enostavnejših a[i] in b[i] ;
c) čeprav problem vsebuje visoko stopnjo potencialne paralelnosti, ta ni izkoriščena. Program deluje nad posameznimi komponentami vektorjev a,b,c, ne pa nad celimi vektorji hkrati. Vzrok je v von-Neumannovi arhitekturi, ki ne omogoča paralelnega množenja istoležnih komponent vektorjev in seštevanja tako dobljenih delnih rezultatov ;
d) konstanta n zmanjšuje uporabnost programa, saj ga lahko uporabimo le nad vektorji dolžine n ;
f) program navaja imena svojih argumentov, s čimer je zmanjšana njegova uporabnost samo na vektorje z imeni a,b,c. Če pa so a,b,c imena formalnih parametrov, nujno sledi uporaba relativno zapletenih mehanizmov prenašanja parametrov ;
g) nekatere operacije so opisane s simboli, ki so porazdeljeni po programu, namesto da bi bile opisane s simbolom na enem mestu.

FP program za računanje skalarnega produkta lahko definiramo kot:

```
DEF SP = (/+)∘(α*)∘Trans
```

Sestavljajo ga tri osnovne funkcije: seštevanje + , množenje * in transpozicija Trans ter tri funkcijske oblike: vstavljanje / , uporaba nad vsemi elementi α in kompozitum ∘ .
Program ima sledeč pomen :



Uporabo (aplikacijo) funkcije f nad argumentom x zapišemo z f:x . Tedaj lahko izvajanje zgornjega programa nad argumentom, ki je par vektorjev <<1,2,3>,<6,5,4>>, opišemo tako:

```
SP:< <1,2,3>,<5,5,4> > =
= (/+) (α*) Trans:< <1,2,3>,<6,5,4> > =
= (/+) (α*):( Trans:< <1,2,3>,<6,5,4> > ) =
= (/+) (α*):< <1,6>,<2,5>,<3,4> > =
= (/+):( (α*):< <1,6>,<2,5>,<3,4> > ) =
= (/+):< *:<1,6>, *:<2,5>, *:<3,4> > =
= (/+):< 6,10,12 > =
= +:< 6, +:< 10,12 > > =
= +:< 6, 22 > )=
= 28
```

Opazimo sledeče lastnosti programa SP :

a) računanje temelji le na dveh pravilih: na pravilu uporabe funkcije nad objektom in pravilu razvoja funkcijske oblike ;
b) program je hierarhičen, kar pomeni, da je sestavljen na podlagi pravil, ki enostavnejše dele programa sestavljajo v kompleksnejše. Tudi ti enostavnejši deli so sestavljeni na podlagi istih pravil, ali pa so elementarni (osnovne funkcije ali objekti);
c) je statičen, kar pomeni, da lahko njegov pomen razumemo že, če ga beremo z desne v levo , seveda ob predpostavki, da poznamo pomen osnovnih funkcij in funkcijskih oblik
d) deluje nad celimi vektorji hkrati
e) uporabimo ga lahko nad poljubno dolgimi vektorji, saj ne vsebuje nobene konstante ;
f) ne poimenuje svojih argumentov , ker deluje le nad njihovimi vrednostmi .

Funkcionalno programirni sistem

FP sistem sestavljajo:

- množica objektov O
- množica F funkcij, ki preslikavajo objekte v objekte
- operator : aplikacije funkcije nad objektom
- množica E funkcijskih oblik, s katerimi sestavljamo obstoječe funkcije in objekte v nove funkcije
- množica D definicij, ki nekaterim funkcijam iz F prirejajo nova imena

Objekt

Objekt je lahko:

- nedefiniran, in ga označimo z ⊥ ;
- atom ;
- zaporedje $\langle x_1, x_2, \ldots, x_n \rangle$, kjer je x_i objekt.

Zaporedje, ki ne vsebuje nobenega elementa označimo z ∅. Zaporedje je nedefinirano, torej enako ⊥ , če vsebuje vsaj en nedefiniran element.
Torej izbrana množica atomov A in pravilo sestavljanja v zaporedje določata množico objektov O. Za množico atomov navadno izberemo množico končnih nizov črk, cifer ali posebnih znakov. Atoma T in F uporabljamo za opis logičnih vrednosti True in False.

Primeri objektov : ⊥ 1.5 ∅ AB3 <AB,1,2.3>
<A,<<1>,C>,D> <A,⊥>

Prvi in zadnja objekta so nedefinirani.

Aplikacija

FP sistem ima samo eno operacijo, ki jo imenujemo aplikacija. Če je f funkcija in x objekt, potem aplikacijo f nad x zapišemo z f:x . Tu je f operator, x pa operand aplikacije f:x . f:x je objekt, ki je rezultat izvršitve funkcije f nad objektom x.

Primeri aplikacije : +:<1,2>=3 2:<A,B,C>=B
tl:<A,B,C>=<B,C>

Funkcije

Vse funkcije preslikavajo objekte v objekte in tudi ohranjajo nedefiniranost, torej velja f:⊥ = ⊥ za vsako funkcijo f .

Funkcija je lahko :

- osnovna (primitivna) ;
- sestavljena ;
- definirana .

Osnovne funkcije podaja že sam FP sistem. Sestavljeno funkcijo dobimo na podlagi obstoječih osnovnih, sestavljenih ali definiranih funkcij, če nad njimi uporabimo eno ali več pravil sestavljanja tj. funkcijskih oblik. Če je funkcija osnovna, je tudi sestavljena. Definirana funkcija je funkcija f, za katero v množici definicij obstaja definicija oblike DEF f ≡ E(f,g,...,h), pri čemer je desni del definicije sestavljena funkcija.
Razlikujemo dva primera, ko je f:x = ⊥ . Če se računanje f:x konča v končno mnogo korakih in vrne vrednost ⊥ , je f nedefinirana pri x. Torej se izračun f nad x konča, vendar ne vrne smiselnega rezultata. Če pa se izračun f nad x ne konča v končno mnogo korakih, zopet velja f:x = ⊥ , vendar tokrat pravimo, da je f pri x neustavljiva .
Žaljeno je, da FP sistem vsebuje široko uporaben nabor osnovnih funkcij. V naslednjih primerih je opisanih nekaj osnovnih funkcij, stroge definicije pa so podane v dodatku A.

'Selekcija' je funkcija i, ki vrne i-ti element danega zaporedja. Če takega elementa ni, vrne vrednost ⊥ . Primeri selekcije:

3:<A,B,C>=C 3:<A,B>=⊥ 1:<<1,2>,3>=<1,2>

'Rep' je funkcija tl, ki izloči prvi element zaporedja in vrne njegov preostanek . Primeri:

tl:<A,B,C>=<B,C> tl:<A>=∅ tl:∅ = ⊥

'Atom' ugotovi, če je dani objekt atom. Primeri atoma: Atom:B=T Atom:<A,B>=F

'Enakost' je funkcija eq, ki ugotavlja enakost dveh elementov zaporedja. Primeri enakosti:

eq:<A,B>=F eq:<<C,D>,<C,D>>=T

'Praznost' je funkcija null, ki ugotovi, če je zaporedje prazno. Primer:

null:∅ =T null: ⊥=⊥ null:<A,B>=F

'Obrat' je funkcija reverse, ki obrne vrstni red elementov zaporedja. Primer obrata:

reverse:<A,B,<C,D>> = <<C,D>,B,A>

'Transpozicija' je funkcija trans, ki istoležne elemente zaporedij združi v nova zaporedja. Primer transpozicije:

```
trans:<<A,B,C>,<D,E,F>,<G,H,J>,<K,L,M>> =
      = <<A,D,G,K>,<B,E,H,L>,<C,F,J,M>>
```

Funkcijske oblike

Funkcijska oblika je izraz, ki označuje funkcijo. Le-ta je odvisna od drugih funkcij ali objektov - parametrov funkcijske oblike. Npr., če sta f in g funkciji, je fog funkcijska oblika, ki ji pravimo kompozitum funkcij f in g. f in g sta parametra te funkcijske oblike. fog je funkcija, katere pomen opišemo takole: (fog):x = f:(g:x). Funkcijske oblike imajo lahko za parametre tudi objekte. če je x objekt, je $\overline{x}$ funkcija, ki nad vsakim definiranim objektom zavzame vrednost x.
V naslednjih primerih je opisanih nekaj funkcijskih oblik, njihove stroge definicije pa so podane v dodatku B.

Primer kompozituma funkcij 'rep' in 'obrat' :

```
  tl∘reverse:<A,B,C> = tl:<C,B,A> = <B,A> .
```

Konstrukcija funkcij f in g je funkcija [f,g]. Funkcijo [f,g] uporabimo nad objektom tako, da nad njim istočasno uporabimo funkciji f in g ter dobljena rezultata združimo v zaporedje. Primer konstrukcije :

```
        [1,2]:<A,B,C> = <<B,C>,B>
```

Levo vstavljanje funkcije f opišemo z /f. Primer levega vstavljanja funkcije + :

```
    /+:<1,2,3>  =  +:<1,/+:<2,3>> =
=   +:<1,+:<2,/+:<3>>>  =  +:<1,+:<2,3>> =
=   +:<1,5>  =  6
```

Istočasno uporabo funkcije f nad vsemi elementi zaporedja opišemo z αf . Primer uporabe funkcije * nad vsemi:

```
α*:<<1,2>,<3,4>> = <*:<1,2>,*:<3,4>> = <2,12>
```

Pogojno izvršitev funkcij f in g glede na funkcijo p opišemo s (p -->f ; g). Rezultat izvršitve funkcije (p -->f ; g) nad objektom x je odvisen od vrednosti p:x. če je le-ta enaka T (True), je rezultat f:x, sicer pa g:x. Primer:

```
  (Atom -->id; reverse) : <A,B,C> = <C,B,A> .
```

Definicije

Definicija v FP sistemu je izraz oblike

```
DEF l <==> r
```

kjer je na levi strani nek še neuporabljen simbol, na desni pa funkcija. Leva stran definicije lahko vstopa kot parameter v desni del, s čimer omogočimo tudi rekurzivne definicije.

Primeri definicij :

```
DEF u <==> [1,tl,2]

Npr. u:<A,B,C> = [1,tl,2]:<A,B,C>= <A,<B,C>,B>

DEF last1 <==> 1∘reverse
```

Funkcija last1 izbere zadnji element zaporedja.
Npr. last1:<1,2,3> = 1∘rev:<1,2,3>=1:<3,2,1>=3

```
DEF last <==> nullotl --> 1 ; lastotl
```

definira isto funkcijo kot last1, tokrat na rekurziven način.

```
last:<A,B> = (nullotl --> 1; lastotl):<A,B>
           = lastotl:<A,B>
           = last:(tl:<A,B>)
           = last:<B>
           = (nullotl --> 1; lastotl):<B>
           = 1:<B>
           = B
```

Lahko se zgodi, da kaka definicija definira funkcijo, ki ni ustavljiva. Množica definicij je dobro oblikovana , če ne vsebuje nobene definicije z enako levo in desno stranjo.

Semantika FP sistema

FP sistem je določen, če poznamo:

- množico atomov A ;
- množico osnovnih funkcij P ;
- množico funkcijskih oblik F ;
- dobro oblikovano množico definicij D .

Za to, da bi poznali semantiko danega FP sistema, moramo za vsako funkcijo f in objekt x vedeti, kako uporabimo f nad x. Obstajajo štiri možnosti :

- f je osnovna funkcija ;
- f je sestavljena ;
- f je definirana ;
- za f ne velja nobena od zgornjih točk .

Uporabo osnovne funkcije pojasnjuje že sam FP sistem. Uporaba sestavljene funkcije temelji na razvoju funkcijske oblike, na podlagi katere je nastala, ter na uporabi parametrov te funkcijske oblike. Definirano funkcijo uporabimo tako, da uporabimo desni del njene definicije. če za f ne velja nobena od zgornjih točk, velja f:x = ⊥ za vsak objekt x.

FP sistemi kot programirni jeziki

FP sisteme lahko smatramo za programirne jezike. Osnovne funkcije in funkcijske oblike so osnovni stavki danega programirnega jezika. Izbira različnih osnovnih funkcij in funkcijskih oblik omogoča izgradnjo programirnih jezikov različnih zmogljivosti. Množica definicij je programska knjižnica.
Funkcija f je program, objekt x je vsebina pomnilnika (vhodni podatek), f:x pa vsebina pomnilnika po izvršitvi programa f nad x.
Dokazovanje pravilnosti programov v konvencionalnih (von Neumannovih) jezikih je zapleteno. Z definiranjem algebre programov, prirejene FP sistemom, je podana osnova za mehanično ugotavljanje in dokazovanje lastnosti programov FP sistema. Algebra programov omogoča tudi uporabniškemu programerju, da ugotavlja in dokazuje lastnosti svojih programov, ne da bi zahtevala od njega pretirano teoretično znanje. Prednost takega dokazovalnega sistema je tudi ta, da programer uporablja pri dokazovanju isti jezik, v katerem so napisani programi. Torej pri dokazovanju lastnosti programov ni potrebno preiti na nek višji, zunanji logični nivo.
Jedro algebre programov sestavlja množica zakonov in izrekov, ki povedo, kdaj je neka funkcija ekvivalentna drugi funkciji.

Spremenljivke algebre programov so funkcije ustreznega FP sistema, operacije algebre pa so funkcijske oblike tega sistema. Tako je npr. [f,g]oh izraz v algebri programov, v katerem so spremenljivke f,g,h poljubne funkcije FP sistema. Zakon v algebri programov ima npr. obliko [f,g]oh <==> [foh,goh]. Ta zakon pravi, da za poljubne funkcije f,g,h sistema velja, da je funkcija na levi strani ekvivalentna funkciji na desni.
Nekaj zakonov algebre programov:

[f,g]oh <==> [foh,goh]

(p-->f;g)oh <==> (poh-->foh;goh)

ho(p-->f;g) <==> (p-->hof;hog)

[f,(p-->g;h)] <==> (p --> [f,g];[f,h])

Dodatek A

Pri naslednjih definicijah uporabljamo izraze

$$p_1 \text{-->} e_1 ; p_2 \text{-->} e_2 ; \ldots p_{k-1} \text{-->} e_{k-1} ; e_k$$

katerih pomen je sledeč:

če p_1 tedaj e_1 sicer pa
če p_2 tedaj e_2 sicer pa
.....
če p_{k-1} tedaj e_{k-1} sicer pa e_k .

V definicijah so x, x_i, y, y_i, z, z_i objekti.

Selekcija

1:x <==> x=<x1,...,xn> --> x1 ; ⊥
in za vsako pozitivno število s:
s:x <==> x=<x1,...,xn> in n≥s --> xs ; ⊥

Rep

tl:x <==> x=<x1> --> ∅ ;
x=<x1,...,xn> in n≥2 --> <x2,...,xn> ; ⊥

Identiteta id:x <==> x

Atom

atom:x <==> x je atom --> T ; x ni ⊥ --> F ; ⊥

Enakost

eq:x <==> x=<y,z> in y=z --> T ;
x=<y,z> in y≠z --> F ; ⊥

Praznost

null:x <==> x=∅ --> T ; x ni ⊥ --> F ; ⊥

Obrat

rev:x <==> x=∅ --> ∅ ;
x=<x1,...,xn> --> <xn,...,x1> ; ⊥

Distribucija z leve

disl:x <==> x=<y,∅> --> ∅ ;
x=<y,<z1,...,zn>> -->
<<y,z1>,...,<y,zn>> ; ⊥

Distribucija z desne

disr:x <==> x=<∅,y> --> ∅ ;
x=<<z1,...,zn>,y> -->
<<z1,y>,...,<zn,y>> ; ⊥

Dolžina

len:x <==> x=<x1,..,xn> --> n ; x=∅ --> 0 ; ⊥

Vsota,Razlika,Produkt,Kvocient

+:x <==> x=<y,z> in y,z števili --> y+z ; ⊥
-:x <==> x=<y,z> in y,z števili --> y-z ; ⊥
*:x <==> x=<y,z> in y,z števili --> y*z ; ⊥
div:x <==> x=<y,z> in y,z števili --> y/z ; ⊥

kjer y/0 = ⊥

Inkrementiranje,Dekrementiranje

ad:x <==> x število --> x+1 ; ⊥
sb:x <==> x število --> x-1 ; ⊥

Transpozicija

trans:x <==> x=<∅,...,∅> --> ∅ ;
x=<x1,...,xn> --> <y1,...,ym> ; ⊥

kjer $x_i = \langle x_{i1}, x_{i2}, \ldots, x_{im} \rangle$ in
$y_j = \langle x_{1j}, x_{2j}, \ldots, x_{nj} \rangle$, $1 \le i \le n$, $1 \le j \le m$

Konjunkcija,Disjunkcija,Negacija

and:x <==> x=<T,T> -->T ; x=<T,F> ali x=<F,T>
ali x=<F,F> --> F ; ⊥
or:x <==> x=<T,T> ali x=<T,F>
ali x=<F,T> --> T ; x=<F,F> --> F ; ⊥
not:x <==> x=<T> --> F ; x=<F> --> T ; ⊥

Levo in desno pripenjanje

apndl:x <==> x=<y,∅> --> <y> ;
x=<y,<z1,...,zn>> --> <y,z1,...,zn> ; ⊥
apndr:x <==> x=<∅,y> --> <y> ;
x=<<z1,...,zn>,y> --> <z1,...,zn,y> ; ⊥

Desna selekcija, Desni rep

1r:x <==> x=<x1,...,xn> --> xn ; ⊥
2r:x <==> x=<x1,...,xn> in n≥2 --> xn-1 ; ⊥
itd.
tlr:x <==> x=<x1> --> ∅ ; x=<x1,...,xn> in
n≥2 --> <x1,...,xn-1>> ; ⊥

Leva in desna rotacija

rotl:x <==> x=∅ --> ∅ ; x=<x1> --> <x1> ;
x=<x1,...,xn> in n≥2 -->
<x2,...,xn,x1> ; ⊥

rotr:x <==> x=∅ --> ∅ ; x=<x1> --> <x1> ;
x=<x1,...,xn> in n≥2 -->
<xn,x1,x2,...,xn-1 > ; ⊥

Dodatek B

Kompozicija (fog):x <==> f:(g:x)

Konstrukcija [f1,...,fn]:x<==><f1:x,...,fn:x>

Pogoj (p --> f; g):x <==> (p:x)=T --> f:x ;
(p:x)=F --> g:x;⊥

Konstanta $\bar{x}$:y <==> y=⊥ --> ⊥ ; x

Levo in desno vstavljanje

/f:x <==> x=<x1> --> x1 ;
x=<x1,...,xn>
in n≥2 --> f:<x1, /f:<x2,...,xn> > ; ⊥

\f:x <==> x=<x1> --> x1 ;
x=<x1,...,x >
in n≥2 --> f:<\f<x1,...,xn-1 >, xn >; ⊥

Uporaba nad vsemi

af:x <==> x=∅ --> ∅ ;
x=<x1,...,xn> --> <f:x1,...,f:xn> ; ⊥

Dvojiško v eniško

(bu f x):y <==> f:<x,y> kjer je x objekt

Zanka

(while p f):x <==> p:x=T-->(while p f):(f:x) ;
p:x=F-->x ; ⊥

Literatura

[1] W.B.Ackerman: 'Data Flow Languages', Computer, Special Issue on Data Flow Systems, Vol.15, No.2, February 1982

[2] J. Backus: 'Can Programming Be Liberated from von Neumann Style? A Functional Style and Its Algebra of Programs', Comm. of the ACM, Vol 21, No.8, August 1978

[3] J. Backus: 'The Algebra of Functional Programs: Function Level Reasoning, Linear Equations and Extended Definitions', Formalization of Programming Concepts, Lecture Notes in Computer Science, 107

[4] J.Šilc,B.Robič,B.Mihovilović: 'Podatkovno vodene računalniške arhitekture', Posvetovanje in seminarji Informatica '85, Nova Gorica, September 1985

PODATKOVNO VODENE RAČUNALNIŠKE ARHITEKTURE

Jurij Šilc, Borut Robič in Branko Mihovilović
Institut 'Jožef Stefan', Ljubljana

UDK: 681.3.014

Računalniške arhitekture lahko razvrstimo glede na podatkovni in krmilni mehanizem. Podatkovni mehanizem določa način, kako se ukazom dodeljujejo argumenti, krmilni mehanizem pa določa vpliv izršitve enega ukaza na izvršitev ostalih ukazov. Glede na podatkovni in krmilni mehanizem razvrstimo računalniške arhitekture v pet kategorij. V podatkovnopretokovnih računalnikih je izvajanje instrukcij podatkovno vodeno; instrukcije postanejo izvršljive takoj, ko so na voljo vrednosti vseh vhodnih argumentov. Podatkovno vodenim arhitekturam pripadajo visoki programirni jeziki z enkratno prireditvijo (Id, Val, Lucid, Valid, SISAL). Materialno opremo podatkovnopretokonih računalnikov sestavljajo procesni, pomnilniški in komunikacijski del, ki so med seboj krožno povezani. Končno je prikazan tudi način, kako lahko uporabimo podatkovno vodeno arhitekturo kot podporo logičnemu programiranju.

DATA-DRIVEN COMPUTER ARHITECTURES - Basically, computer architectures share two fundamental mechanisms, these are data and control mechanisms. The data mechanism defines the way a particular argument is communicated by a number of instructions. The control mechanism defines how one instruction causes the execution of one or more others. In each category of computer, data mechanisms and control mechanisms are largely incompatible sets of alternative concepts. In data flow computer, instruction execution is data driven; the availability of input operands triggers the execution of the instruction that consumes the inputs. Data flow computers are programmed in very high level single-assignment languages (Id, Val, Lucid, Valid, SISAL). The packet communication machine organization is the most obvious for supporting the data flow concepts. It consists of a circular instruction execution pipeline of resources in which processors, communications, and memories are interspersed with 'pools of work' containing different types of information packets. In this paper we also consider a data-driven model for interpreting logic program and investigate the architectural requirements necessary to support its implementation. It will be shown that the model is capable of exploiting the capabilities of highly - parallel dataflow architectures.

§ 1. UVOD

Razvoj računalništva v bodoče je odvisen predvsem od uspešnosti raziskav na naslednjih področjih:

* Umetna inteligenca;
  - metodologije, ki izražajo in povzemajo znanje,
  - uporabniku prirejeni I/O v obliki naravnih jezikov, govora in slik.
* Programirni inženiring;
  - novi visoki programirni jeziki in računski postopki,
  - programska orodja zgrajena na sistemih, kot npr. Unix.
* Računalniške arhitekture;
  - razpršene arhitekture, kot podpora računalniškim mrežam,
  - paralelne arhitekture za zelo hitre računalnike,
  - VLSI arhitekture z največjo izrabo potenciala VLSI tehnologije.
* VLSI tehnologija;
  - VLSI CAD sistemi,
  - nove tehnologije, npr. Ga-As, Josephsonov spoj,...

Tako tehnološko, kakor tudi sociološko smo pripravljeni na novo - peto računalniško generacijo. Računalniki so prešli od znanstveno usmerjenih aplikacij v 50-tih letih, preko komercialno industrijskih aplikacij v 60-tih in 70-tih letih, v inteligentno široko potrošniško elektroniko v 80-tih in 90-tih letih.

§ 2. PREGLED ARHITEKTUR

Sodobne arhitekture imajo za cilj: povečati računalniško moč sodobnih računalniških sistemov, kar pomeni, povečati njihovo zmogljivost in hitrost procesiranja in odpraviti ozka grla v procesu računanja, ki so posledica klasične arhitekture. V novi generaciji splošno namenskih računalnikov bo procesiranje prešlo iz sekvenčnega, centraliziranega sveta v paralelni, decentralizirani svet procesiranja. To je zanesljivo, toda kakšni računalniki bodo prevladovali ni popolnoma jasno. Raziskovalci prisegajo na več mogočih realizacij nove generacije:

* 'Peta generacija' računalnikov;
  - knowledge information processing sistemi,
  - podpora ekspertnim sistemom,
  - uporabniku prilagojeni I/O.
* Superračunalniki;
  - velika izraba paralelizma,
  - uporabnost za obsežna numerična izračunavanja.
* Arhitekture z VLSI (ULSI) procesorji;
  - multi mikroprocesorski sistemi,
  - posebno namenski sistemi z maksimalno izkoriščenim potencialom VLSI tehnologije (1M tranzistorjev/čip).
* Integracija komunikacij in računalnikov;
  - popolnoma integrirana računalniško komunikacijska mreža,
  - velika omrežja, lokalne mreže, paralelne arhitekture.

KRMILNI MEHANIZEM	PODATKOVNI MEHANIZEM: dodeljevanje pomnilnika	PODATKOVNI MEHANIZEM: prehajanje sporočil
ukazno vodenje	von Neumannova arhitektura proceduralni jeziki	
podatkovno vodenje		podatkovno pretokovna arhitektura jeziki z enkratno prireditvijo
vodenje na zahtevo	redukcijska arhitektura aplikacijski oz. funkcionalni jeziki	
vzorčno vodenje	logična arhitektura logični jeziki	objektno orientirana arhitektura objektno usmerjeni jeziki

Slika 1. - Pregled računalniških arhitektur.

Računalniške arhitekture delimo glede na podatkovni in krmilni mehanizem.

* Podatkovni mehanizem določa način, kako se dodeljujejo ukazom zahtevani argumenti. Ločimo dva osnovna načina:
  - Dodeljevanje pomnilnika; vsakemu ukazu se dodelijo deli pomnilnika, v katerih so argumenti.
  - Prehajanje sporočil; argumenti se prenašajo v obliki sporočila (paketa) od ukaza do ukaza.
* Krmilni mehanizem določa vpliv izvršitve enega ukaza na izvrševanje ostalih ukazov. Ločimo:
  - Ukazno vodenje; ukaz se izvrši, ko je selektiran s krmilno strukturo.
  - Podatkovno vodenje; ukaz se izvrši natanko tedaj, ko so prisotni vsi vhodni argumenti.
  - Vodenje na zahtevo; ukaz se izvrši, ko zazna zahtevo drugega ukaza po njegovem rezultatu.
  - Vzorčno vodenje; vzorec, ki omogoča izvršitev ukaza se primerja z nekim vzorcem in v primeru ujemanja dovoli izvršitev.

Glede na podatkovni in krmilni mehanizem razvrstimo računalniške arhitekture v pet kategorij, katerim pripadajo tudi ustrezni programirni jeziki [1] (slika 1):

* Von Neumannove arhitekture, ki so ukazno vodene in uporabljajo dodeljevanje pomnilnika; pripadajo jim proceduralni jeziki, kot so Basic, Pascal, Fortran, ...
* Redukcijske arhitekture, katerih krmilni mehanizem temelji na načelih vodenja na zahtevo, podatkovni mehanizem pa je bodisi dodeljevanje pomnilnika ali prehajanje sporočil; takšne arhitekture uporabljajo aplikacijske oz. funkcionalne jezike, kot so čisti Lisp, SASL in FP.
* Podatkovno pretokovne arhitekture, ki so podatkovno vodene in imajo dodeljevanje argumentov s prehajanjem sporočil; uporabljajo jezike z enkratno prireditvijo, kakršni so Val, Id, Valid, Lucid, SISAL.
* Objektno orientirane arhitekture, ki so vzorčno vodene in uporabljajo mehanizem prehajanja sporočil; tem arhitekturam pripadajo objektno usmerjeni jeziki, kakršen je npr. Small-talk.
* Logične arhitekture, ki so tudi vzorčno vodene in temeljijo na dodeljevanju pomnilnika; pripadajo jim predikatno logični programirni jeziki, kakršen je Prolog.

Najenostavnejša je von Neumannova arhitektura, kjer natančno opišemo kako dobimo želeni rezultat. Pri kompleksnejših, kot je npr. logična arhitektura, pa navedemo le, kaj je želeni rezultat.

§ 3. PODATKOVNO VODENE ARHITEKTURE

V von Neumannovih arhitekturah je potek izvajanja ukazov (vnaprej) podan eksplicitno. Takšne arhitekture imajo dve značilnosti:

- Prvič, imajo globalni naslovljivi pomnilnik, ki pomni programe ter podatke in katerega vsebina se v skladu s programskimi instrukcijami pogosto ažurira.
- In drugič, vsebujejo programski števec, katerega vsebina je naslov naslednje instrukcije, ki naj se izvrši, torej je s tem določena sekvenca instrukcij, ki se izvajajo (ukazno vodenje).

Takšno vodenje poteka programa je temeljna omejitev teh arhitektur, še posebno pri vzporednem procesiranju.

jezik z enkratno prireditvijo
(Val, Id, Valid, Lucid, SISAL ...)
↓
podatkovno pretokovni programski graf
↓
podatkovno vodena arhitektura
(VLSI imlementacija)

Slika 2. - Na jeziku zasnovana podatkovno vodena arhitektura.

Kakšne so alternative? Ena od mogočih je koncept podatkovnega vodenja, torej podatkovno pretokovna arhitektura. Podatkovno pretokovni računalniki nimajo nobene od prej omenjenih von Neumannovih značilnosti. Njihova struktura je zasnovana tako, da poteka procesiranje na osnovi vrednosti in ne naslovov spremenljivk. Ti sistemi temeljijo na asinhronosti in funkcionalnosti. Vse operacije oz. ukazi so funkcije, ki postanejo izvršljive takoj, ko so na voljo vrednosti vseh vhodnih argumentov. Ukazi, ki postanejo izvršljivi, se lahko izvršijo sočasno [2].

S pravilno izbiro programske (programirni jeziki) in materialne opreme, je mogoče s takšno arhitekturo izkoristiti inherentni paralelizem v največji mogoči meri. Razen maksimalnega izkoriščanja inherentne paralelnosti v algoritmu, imajo računalniki vodeni s pretokom podatkov sposobnost dinamičnega prilagajanja svoje strukture strukturi algoritma. V bistvu smatramo te arhitekture kot arhitekture, zasnovane na jeziku, kjer programski graf - 'strojni jezik' - predstavlja medstopnjo med visokim programirnim jezikom in arhitekturo (slika 2). Od teh arhitektur se zahteva učinkovita implementacija programskega grafa.

§ 3.1. Jeziki z enkratno prireditvijo

Hkrati z izboljšavami von Neumannovih arhitektur so se pojavili tudi nekateri novi visoki programirni jeziki, ki so vsebovali sintaksne konstrukte, s pomočjo katerih so postale izboljšane arhitekturne lastnosti vidne programerju. Večina teh jezikov je nenaravnih v smislu, da v preveliki meri odražajo arhitekturo, manj pa način na katerega mora programer razmišljati pri reševanju problema.

Tako kot ostale oblike paralelnih računalnikov zahtevajo tudi podatkovno pretokovni računalniki (zaradi čimboljše izrabe potencialnih arhitekturnih lastnosti) posebne visoke programirne jezike, t.i. jezike z enkratno prireditvijo ali podatkovno pretokovne jezike. Ti jeziki, za razliko od konvencionalnih jezikov, ne poznajo stranskih učinkov. Naslednja lastnost teh jezikov je lokalnost učinka, kar pomeni, da ukazi nimajo nepotrebnih , daleč segajočih podatkovnih odvisnosti. Nadalje, ti jeziki uporabljajo tudi pravilo o enkratni prireditvi [2].

Jeziki z enkratno prireditvijo imajo v sintaksnem smislu veliko skupnih lastnosti s proceduralnimi jeziki, saj uporabljajo podobne programske konstrukte. Za razliko od proceduralnih jezikov, pri katerih identifikator predstavlja naslovljivi del pomnilnika, pa pri jezikih z enkratno prireditvijo predstavlja povezavo v nekem grafu. Operacija pa predstavlja točko tega grafa. Bistveno pa se ti jeziki razlikujejo po semantiki. Prireditev v proceduralnih jezikih pomeni prenos vrednosti argumentov iz ustreznih delov pomnilnika v registre, izvršitev operacije in prenos rezultatov v ustrezno lokacijo v pomnilnik. Pri jezikih z enkratno prireditvijo pa prireditev pomeni izvršitev operacije, ki se nahaja v točki grafa, nad vhodnimi podatki, ki gredo v to točko in vpis rezultata v izhodno povezavo, ki gre iz te točke grafa.

Oglejmo si sedaj kaj je in zakaj je potrebna enkratna prireditev. Za ilustracijo vzemimo prireditev x := a + b v nekem proceduralnem jeziku. Stavek moramo razumeti kot:
- vzemi vrednosti iz pomnilniških lokacij z naslovoma a in b,
- izvrši operacijo + in
- prenesi rezultat te operacije v pomnilniško lokacijo z naslovom x.

Vidimo, da zgornjemu stavku lahko sledi poljubno mnogo stavkov, ki imajo na levi strani x; stavki pač spreminjajo vsebino lokacije z naslovom x. Sedaj pa si oglejmo sintaktično enako prireditev x := a + b v jeziku z enkratno prireditvijo. Stavek moramo tu razumeti takole:
- obstaja točka grafa, ki je nosilec operacije + in v katero se stekata povezavi z imenoma a in b (v grafu ima lahko samo ena povezava ime a oz. b!),
- obstaja natanko ena povezava z imenom x in to je ravno povezava, ki gre iz te točke,
- po izvrsitvi operacije +, v povezavah a in b ni več vrednosti, v povezavi x pa je vrednost vsote.

če temu prireditvenemu stavku sledi prireditev, ki ima na levi x, npr x := c - d, potem bi morala povezava x izhajati tudi iz točke, ki nosi operacijo - in ima vhodni povezavi c in d. Torej bi povezava potekala iz dveh točk - taki grafi pa niso dopustni.

Primer jezika z enkratno prireditvijo naj nam ilustrira program za izračun integrala pod krivuljo y=x**2, na intervalu [0,1], ki ga izračunamo s pomočjo trapezoidne aproksimacije s konstantnim korakom 0.02; program je zapisan v jeziku SISAL [3] (slika 3).

```
export Integrate
function Integrate (returns real)
for initial
  int := 0.0;
  y := 0.0;
  x := 0.02
while
  x < 1.0
repeat
  int := 0.01 * (old y + y);
  y := old x * old x;
  x := old x + 0.02
returns
  value of sum int
end for
end function
```

Slika 3. - Numerični izračun integrala v jeziku SISAL.

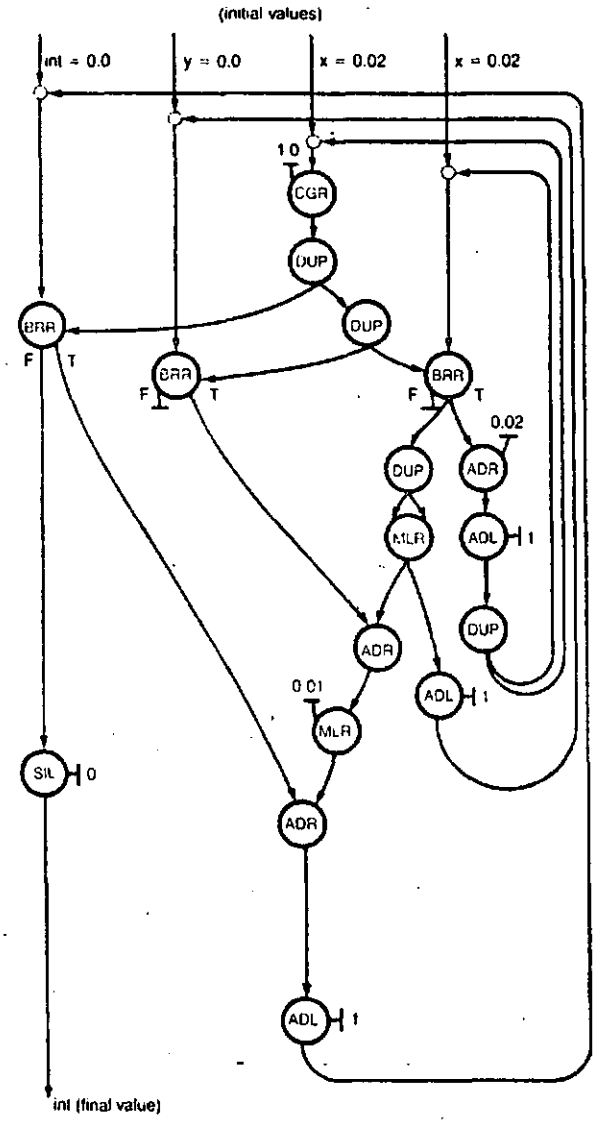
§ 3.2. Programski grafi

Programe krmiljene s pretokom podatkov opisujemo z usmerjenimi grafi, ki jih imenujemo podatkovno pretokovni programski grafi ali krajše programski grafi. Programski grafi so rezultat prevajanja programa zapisanega v enem od visokih podatkovno pretokovnih programirnih jezikov in so torej strojni jezik podatkovno vodenih arhitektur. Prevajanje običajno poteka preko vmesnega zbirnega jezika. Primer takšnega zbirnika prikazuje slika 4.

```
! initialize the loop variables
int = (Data "R 0.0");
x   = (Data "R 0.0");
y   = (Data "R 0.02");
! merge the initial values with the loop
! output values
int_mrg  = (Mer int new_int);
y_mrg    = (Mer y new_y);
x_mrg    = (Mer x new_x);
! test the termination loop
test     = (CGR "R 1.0" x_mrg);
! gate the loop variables into new loop
! instance or direct result to output
gate_int = (BRR int_mrg test);
old_int  = gate_int.R;
old_y    = (BRR y_mrg test).R;
old_x    = (BRR x_mrg test).R;
result   = (SIL gate_int.L "0 0").L;
! loop body: fom new values for loop
!            variables
incr_x   = (ADR old_x "R 0.02");
x_sq     = (MLR old_x old_x);
height_2 = (ADR old_y x_sq);
area     = (MLR "R 0.001" height_2);
cum_area = (ADR old_int area);
!          : increment iterational level
!            for new loop variables
new_int  = (ADL cum_area "I 1").L;
new_y    = (ADL x_sq "I 1").L;
new_x    = (ADL incr_x "I 1").L;
! output the final value of int
           (OPT result "G 0");
```

Slika 4. - Program preveden iz jezika SISAL v zbirnik TASS.

Elementi programskega grafa ponazarjajo pretok podatkov med posameznimi ukazi. Točke grafa ponazarjajo ukaze, usmerjene povezave pa so nosilke vrednosti argumentov ter običajno še dodatnih informacij o delih izračunov katerim pripadajo argumenti. Torej so vhodne povezave nosilke operandov ukaza, ki ga točka predstavlja, izhodne povezave pa hranijo parcialne rezultate. Primer programskega grafa je prikazan na sliki 5.



Slika 5. - Podatkovno pretokovni programski graf.

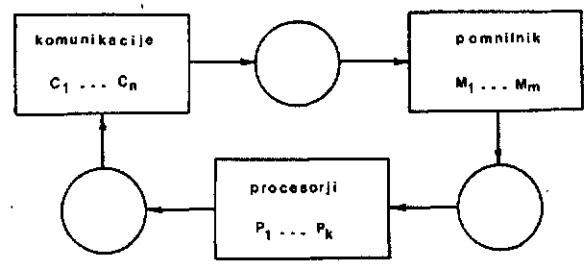
Programski graf je asinhron, kar pomeni, da postanejo ukazi izvršljivi tedaj, ko imajo na voljo vse vhodne operande. Gibanje podatkov skozi programski graf pomeni izvajanje programa. Omeniti velja še eno zelo pomembno lastnost programskih grafov in jasno s tem tudi podatkovno vodenih arhitektur; namreč, ko je operacija oz. ukaz izvršen, vhodni operandi oz. njihove vrednosti, niso več razpoložljivi. Pri cikličnih programskih grafih se pojavijo dodatne težave; v določenih vejah grafa se lahko začno akumulirati vrednosti operandov, kar pomeni, da bi postal graf nedeterminističen. Torej ni možno s prisotnostjo katerekoli vrednosti vhodnega operanda deklarirati točko kot izvršljivo, saj lahko pripadajo operandi popolnoma različnim delom izračuna. Obstaja nekaj rešitev nastalega problema:

* Ciklične programske grafe transformiramo v aciklične. Ta rešitev je zelo nerodna v primeru, ko imamo zanke, katerih pogoj za izstop iz zanke se izračuna šele v času samega izvajanja. V teh primerih je potrebno dinamično generiranje koda.
* V vejah dovoljujemo prisotnost le ene vrednosti. To izvedemo s pomočjo dodatnih povratnih povezav, ki nosijo potrditvene signale, kateri dovoljujejo ustvarjanje novih vrednosti. Z uvedbo teh povezav se promet v grafu podvoji.
* Naslednja rešitev je, da so povezave nosilke paketov, ki poleg vrednosti operandov nosijo še dodatne informacije o delih izračunov, katerim te vrednosti pripadajo. Te dodatne labele običajno imenujemo barve in točka postane izvršljiva, ko so v vseh vhodnih povezavah prisotni paketi enake barve.
* Obstaja še ena možna rešitev; povezave lahko opravljajo funkcijo vrste, kar pomeni, da so vrednosti razvrščene tako, kot so v povezavo prihajale.

Če povzamemo, programski graf je asinhron in determinističen, vse točke imajo pomen funkcij in vrednosti operandov po uporabi niso več razpoložljive [4].

§ 3.3. Materialna oprema

Materialna oprema podatkovno vodenih arhitektur mora biti zasnovana tako, da omogoča učinkovito implementacijo programskega grafa. Tak sistem sestavljajo procesni, pomnilniški in komunikacijski del, ki so med seboj krožno povezani (slika 6).



Slika 6. - Podatkovno vodena arhitektura.

Podatkovno pretokovni računalniki nimajo centralnega procesorja, temveč imajo procesni del, ki ga sestavlja množica nekaj deset, sto ali tisoč procesnih enot. Nimajo niti pomnilnika, kakršnega uporabljajo von Neumannove arhitekture, zato pa imajo pomnilniški del, ki ima potencialno veliko število pomnilniških celic, v katerih se nahajajo vsi podatki o programskem grafu. Za podatkovno pretokovne računalnike je značilno tudi to, da ne uporabljajo sinhronizacijske ure, programskega števca in registrov. Zato pa ima arbitražno vezje, ki usmerja izhode iz celic pomnilniškega dela v ustrezne procesne enote procesnega dela in distribucijsko vezje, ki povezuje procesne enote s pomnilniškimi celicami.

Osnovne ideje podatkovno vodenih arhitektur segajo v pozna šestdeseta leta, ki pa so ob možnosti VLSI implementacije postale ponovno zelo aktualne. VLSI tehnologija, ki omogoča izdelavo mikrovezij s celo 100 pini, je potrebna, saj bi tisoč takšnih vezij, ki bi opravljalo funkcijo usmerjanja in povezovanja, omogočilo uporabo 512 procesorskega podatkovno pretokovnega sistema.

§ 4. LOGIČNO PROGRAMIRANJE NA PODATKOVNO VODENIH ARHITEKTURAH

Logično programiranje je matematični formalizem, s katerim lahko rešujemo probleme na neproceduralni način. Logični programi niso vezani na von Neumannove - sekvenčne arhitekture in so zato primerni za izvajanje na paralelnih arhitekturah. Zato se samo po sebi postavlja vprašanje, kako je mogoče za njihovo izvajanje uporabiti podatkovno vodene arhitekture? S tem bi namreč dosegli, da bi te arhitekture podpirale tudi Prolog, ki je izbran za jezik 'pete' generacije. Odgovor je: narediti prevajalnik, ki bi logični program prevedel v programski graf, to je strojni jezik podatkovno pretokovnega računalnika [5].

Prolog je nepostopkovni ali deklerativni jezik, ki temelji na predikatnem računu prvega reda, ki tudi tvori njegovo sintakso. Omogoča enostavno definiranje relacij med podatki in pa strukturiranje podatkov.

§ 4.1. Logični program

Logični program je množica stavkov, ki imajo obliko

$$p_0 \;:-\; p_1,\ldots,p_m.$$

Vsak p_i, imenovan cilj (literal), ima obliko $p(t_1,\ldots,t_n)$, kjer je p predikatni simbol, $t_1,\ldots,t_n$ pa so elementarni podatkovni objekti (termi), ki so lahko konstante, spremenljivke ali funktorji. Privzemimo omejitev, da so vsi predikati binarne oblike, torej $p(t_j,t_k)$. S to omejitvijo ne izgubimo ničesar na splošnosti, saj lahko vsako n-mestno relacijo pretvorimo v n+1 binarnih relacij. Zaradi enostavnosti se omejimo le na elementarne podatkovne oblike tipov konstanta in spremenljivka. p_0 se imenuje glava stavka, p_1 do p_m pa tvorijo telo stavka. Stavek brez ciljev je enotin stavek, ki izraža eksplicitno dejstvo, to je brezpogojno trditev oz. dejstvo. V primeru programa s slike 7 so stavki, označeni z (1) do (5), ki podajajo relacije "vodja", "raziskovalec" in "programer" med osebami in skupinami, enotini stavki.

```
(1)  vodja(sk1,peter).
(2)  vodja(sk2,gorazd).
(3)  razis(sk1,borut).
(4)  razis(sk1,jure).
(5)  prog(sk2,jure).
                .
                .
                .
(6)  sodel(S,I) :- prog(S,I).
(7)  sodel(S,I) :- razis(S,I).
(8)  nad(I1,I2) :- sodel(S,I1),vodja(S,I2).
                .
                .
                .
(9)  ?- nad(jure,I2).
```

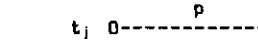
Slika 7. - Primer logičnega programa.

Stavki, ki imajo glavo in telo prinašajo implicitno informacijo in se imenujejo splošni stavki. Takšni so stavki (6) do (8) v primeru s slike 7. Stavka (6) in (7) opisujeta dejstvo, da oseba I, ki je "programer" ali "raziskovalec" skupine S, tudi "sodelavec" te skupine. Stavek (8) pravi, da je oseba I2 v relaciji "vodja" z osebo I1, če je I2 "vodja" skupine S, katere "sodelavec" - "programer" ali "raziskovalec" - je I1.

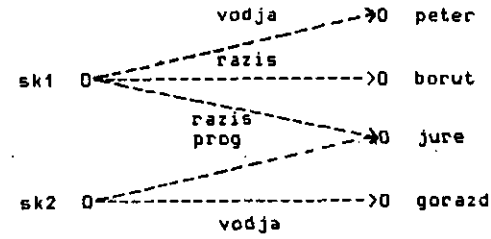
Stavki, ki nimajo glave, so vprašalni stavki. Sistem poišče odgovor nanje tako, da cilj vprašalnega stavka prilagodi na glavo enega izmed splošnih stavkov. Na sliki 7 je vprašalni stavek označen z (9), ki sprašuje "Kdo je Juretu nadrejen?". Cilj tega stavka se prilagodi z glavo stavka (8). Spremenljivka I1 v celem stavku pri tem zavzame vrednost jure. Telo stavka (8) sestavljata cilja sodel(S,jure) in vodja(S,I2). V naslednjem koraku se sodel(S,jure) prilagodi eni od glav stavkov (6) ali (7), npr.(6). Postopek se nadaljuje dokler ni cilj vprašalnega stavka izpolnjen. Kadar cilja ni več mogoče prilagoditi nobeni glavi, pride do vračanja in ponovnega prilagajanja z drugimi glavami ; npr. cilj sodel(S, jure) se prilagodi glavi stavka (7). Če niti prilagajanje niti vračanje ni več mogoče, potem cilja ni mogoče izpolniti.

§ 4.2. Predstavitev logičnega programa s podatkovno pretokovnim grafom

Kot rečeno, logični program sestavljajo dejstva in splošni stavki. Vsako dejstvo vsebuje ime predikata p, ki mu sledita dva elementarna podatkovna objekta t_j in t_k. Množici vseh dejstev logičnega programa priredimo graf tako, da predstavimo vsako dejstvo $p(t_j,t_k)$ s podgrafom oblike

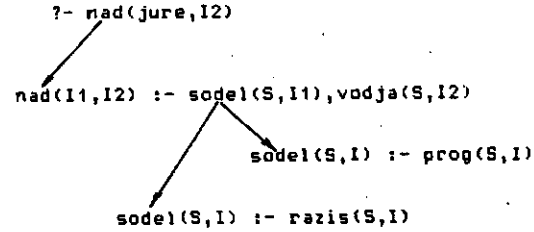


Vozlišča grafa so elementarni podatkovni objekti. Usmerjena povezava iz vozlišča t v t obstaja in nosi oznako p natanko tedaj, ko je p(t ,t) dejstvo. Dobljeni graf imenujemo graf dejstev. Dejstva opisujejo relacije med elementarnimi podatkovnimi objekti, ki v splošnem niso simetrične. Zato je graf dejstev usmerjen. Za primer s slike 7 je graf dejstev prikazan na sliki 8.



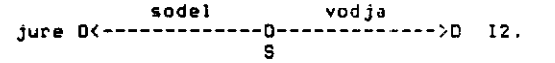
Slika 8. - Graf dejstev.

Splošni stavki, ki imajo glavo in telo, se med seboj povezujejo s kazalci v usmerjeno strukturo na sledeč način: cilj v telesu nekega stavka kaže na vse stavke, katerih glave se z njim ujemajo. Takšno, včasih celo ciklično, strukturo imenujemo strukturo ciljev. Nek cilj L te strukture rešimo tako, da v grafu dejstev poiščemo dejstvo, ki se ujema s tem ciljem ali pa (če to ni možno) rešimo vsaj enega od stavkov, na katere kaže ta cilj. Slika 9 prikazuje strukturo ciljev za primer logičnega programa s slike 7.

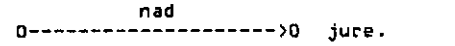


Slika 9. - Struktura ciljev.

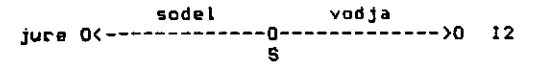
Telesa stavkov lahko predstavimo z grafi na podoben način kot predstavimo dejstva, le da so tu elemetarni podatkovni objekti lahko tudi proste spremenljivke. Tako predstavljene stavke imenujemo graf vzorcev ali kar vzorec. Na primer, v strukturi ciljev na sliki 9 je v stavku v drugi vrsti spremenljivka I1 zaradi prilagoditve na cilj v prvi vrstici vezana na vrednost jure, toda spremenljivki S in I2 sta prosti; temu stavku zato ustreza naslednji vzorec



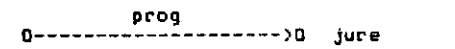
S tem, ko smo logični program prevedli v grafno obliko, postane izvajanje programa ekvivalentno problemu prilagajanja grafa. Potek je naslednji: vzorec se reši tako, da se v grafu dejstev poišče dejstvo, ki se ujema z vzorcem. Npr. vprašalni stavek v našem primeru je vzorec



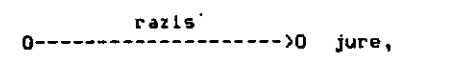
V grafu dejstev poizkusimo poiskati povezavo z oznako "nadrejeni" in točko "jure". Skratka vzorec poizkušamo prilagoditi dejstvu. To prilagajanje ne uspe, ker v strukturi ciljev ni povezave "nadrejeni". Istočasno se pričnе prilagajati tudi vzorec



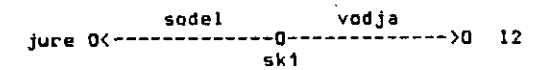
na katerega kaže ciljni vzorec. Cilj "sodelavec" kaže na naslednja dva vzorca



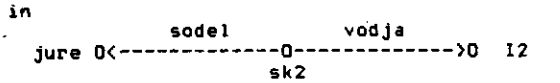
in



ki se tudi sočasno prišneta prilagajati grafu dejstev. Tu prilagajanje uspe. Spremenljivka S se veže na vrednosti "sk1" in "sk2". Po ponovnem prilagajanju vzorcev



in



dobimo iskani rešitvi, ki sta "peter" in "gorazd".

Graf dejstev torej ustreza podatkovno pretokovnemu grafu oz. strojnemu jeziku podatkovno vodene arhitekture. Vsaka točka grafa je logično gledano ativni element, sposoben izmenjave paketov podatkov med točkami. Paketi vsebujejo vzorce (vključno s kazalci), ki se poizkušajo prilagoditi grafu dejstev.

§ 5. ZAKLJUČEK

Večji projekti na področju podakovno vodenih sistemov potekajo:

* Na MIT, kjer sta dve skupini; prva pod vodstvom J. Dennisa razvija sistem z rezinastimi procesorji in druga po vodstvom Arvida, ki gradi VLSI 64 procesorski sistem.
* Na univerzi Utah deluje skupina, ki jo vodi Davis.
* Na CERT v Toulousu dela skupina pod vodstvom J. C. Syra na projektu LAU.
* Na univerzi v Manchestru gradijo eksperimentalni podatkovno vodeni multiprocesorski sistem pod vodstvom J. Gurda in I. Watsuna.
* Na univerzi Tokyo, kjer gradijo računalnik Topstar pod vodstvom T. Suzukija in J. Mot... oke.

Analize učinkovitosti, ki so jih izvršili na realiziranih sistemih so pokazale občutno časovno izboljšanje. Ti sistemi so šele v razvoju, zato obstaja tudi nekaj nerešenih problemov, kot npr. nerešen problem vhodno/izhodnih aktivnosti. Vsekakor pa postajajo podatkovno vodene arhitekture tudi komercialno dosegljive. Pri firmi NEC so izdelali, tako lahko beremo v lanski oktoberski številki revije Computer Design, prvi VLSI čip µPD7281, ki uporablja podatkovno vodeno arhitekturo in je namenjen procesiranju slik [6].

V referatu smo predstavili eno od mogočih arhitektur pete generacije, ki bo morda prevladala na področju superračunalnikov. Takšne arhitekture imajo za cilj povečati računalniško moč sistemov, kar pomeni povečati njihovo zmogljivost in hitrost procesiranja ter odpraviti ozka grla v procesu računanja, ki so posledica von Neumannovega ozkega grla. Večje računalniške zmogljivosti potrebujemo zaradi naraslih potreb pri reševanju problemov na področju umetne inteligence, obdelave slik, razpoznavanja govora, napovedovanja vremenskih situacij, avtomatskega prevajanja jezika, ipd. Obdelava in reševanje takšnih in podobnih problemov s sprejemljivo hitrostjo je mogoča le ob uporabi novih paralelnih - decentraliziranih računalniških arhitektur.

§ 6. LITERATURA

[1] Treleaven P.C., Isabel Gouveira Lima: 'Future Computers: Logic, Data Flow, ..., Control Flow?', Computer, Vol.17, No.4, Mar. 1984, pp.47-57

[2] Computer, Special Issue on Data Flow Systems, Vol.15, No.2, Feb. 1982

[3] Gurd J.R., C.C. Kirkham & I. Watson: 'The Manchester Prototype Dataflow Computer', Comm. of the ACM, Vol.28, No.1, Jan. 1985, pp.34-52

[4] Silc J., B. Robič: 'Osnovna načela DF sistemov', Informatica, št.2, 1985, str.10-15

[5] Bic L.: 'Execution of Logic Programs on a Dataflow Architecture', The 11th Annual International Symposium on Computer Architecture, Ann Arbor, Michigan, June 1984, pp.290-296

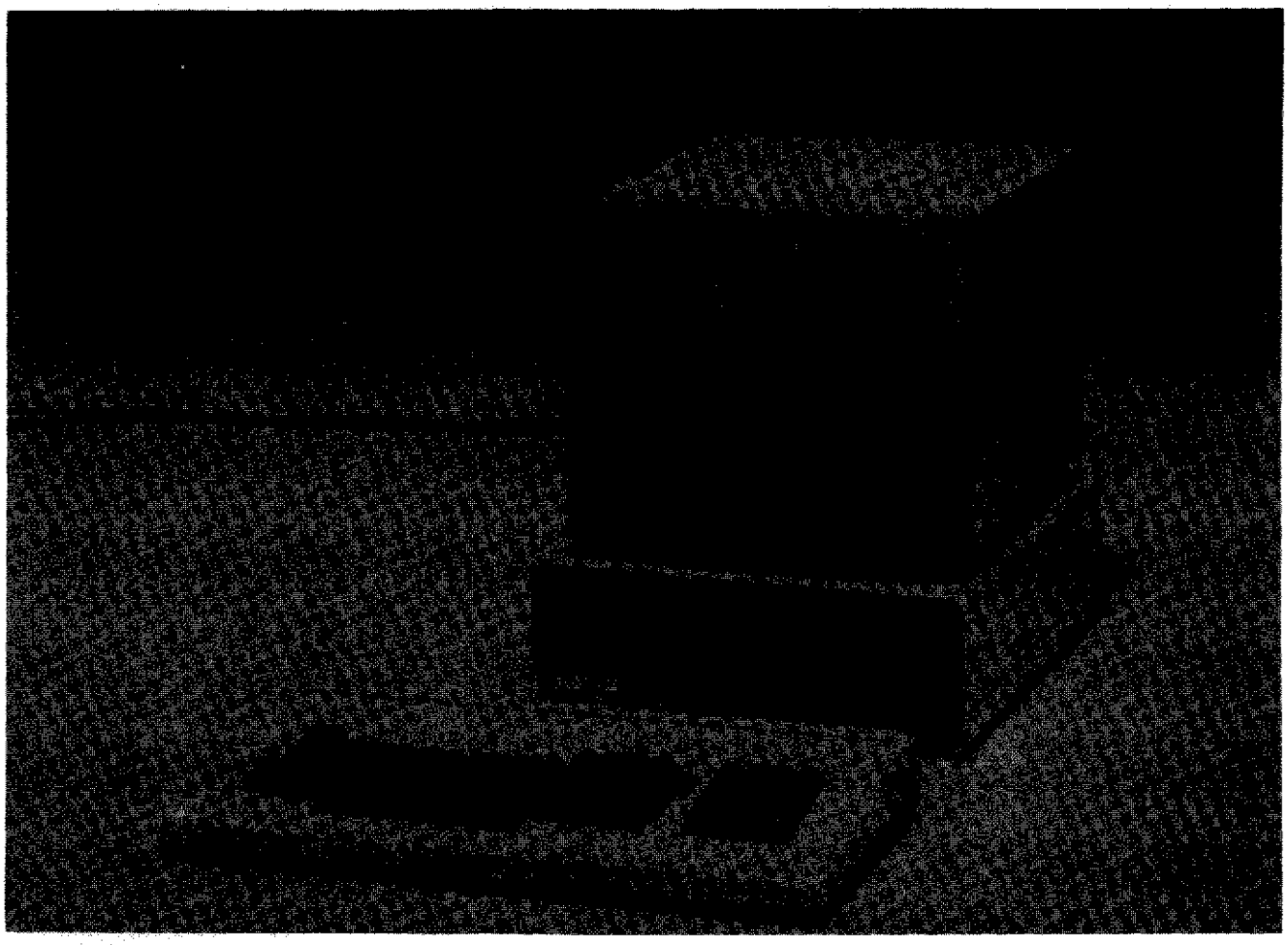
[6] Chong Y. M.: 'Data Flow Chip Optimizes Image Processing', Computer Design, Oct.15 1984, pp.97-103

DIALOG P DIALOG P DIALOG P

Dialog P je osebni računalnik sistemsko odprte zasnove. Operacijski sistem je kompatibilen s CP/M operacijskim sistemom. Njegova uporaba je zelo široka:
poslovna, procesna, laboratorijska in kot pripomoček pri izobraževanju.

Tehnični podatki:

Centralna enota:	procesor Z80B 64 k DRAM pomnilnik 32 k bralni pomnilnik
Disketni pogon:	1 x TEAC 55F, kapaciteta 800 k-byte opcija: 2 x disketni pogon
Tipkovnica:	dodaten numerični del, domač nabor znakov
Monitor:	profesionalni, monokromni, zeleni fosfor P 31
Priključki:	izhod za monitor in za TV sprejemnik, serijski vmesnik RS 232C, sistemsko vodilo
Programska oprema:	FEBASIC, možna uporaba vseh programskih paketov za operacijski sistem CP/M (WORDSTAR, TURBO PASCAL, DBASE II . . .)



gorenje procesna oprema

n. sol. o., 63320 Titovo Velenje, Partizanska cesta 12
Telefon: (063) 853 321
Telegram: GorenjeProcesna
Telex: 33616, 33547 yu sogor