JPRS-UCC-89-001
14 MARCH 1989

FOREIGN
BROADCAST
INFORMATION
SERVICE

JPRS Report

Science & Technology

USSR: Computers

CONTROL SYSTEMS AND MACHINES

JPRS-UCC-89-001

14 MARCH 1989

SCIENCE & TECHNOLOGY

USSR: COMPUTERS

CONTROL SYSTEMS AND MACHINES

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian No 3, May-Jun 1988, pp 3-123

CONTENTS

UDC 681.3

Fault-Tolerant Non-Positional Processors

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian (manuscript received 08 Aug 86; after revision 29 Jan 88) No 3, May-Jun 1988, pp. 3-7

[Article by N. I. Chervyakov]

[Text] Computers are increasingly used in control systems and processes, making the problem of assuring their operating reliability an important one. It is impossible in principle to create absolutely reliable processor components. The processors must assure reliable functioning even with faults and failures in the equipment over a long period of time. This problem can be solved by fault-tolerant processors, capable of continuing to function even when various component failures occur. Fault-tolerant processors are two to three orders of magnitude more reliable than are ordinary processors[1].

Extensive theoretical research and practical development has been conducted in the area of fault-tolerant processors both in our country and abroad.

Actual fault-tolerant systems have been manufactured, including a number of multi-machine and multiprocessor ES and SM computer systems [2], the ESS processors made by *Bell Systems, Pluribus, Stratus–32*[3], etc.

Redundant fault-tolerant systems include systems with multiple redundancy, structural redundancy, hybrid redundancy and gradual degradation[3]. In systems with gradual degradation, tasks are executed using the remaining nondefective modules.

Very promising for the generation of systems with gradual degradation is residue arithmetic, which supports parallel, modular processor structure and has allowed a new approach to the problems of creating a fault-tolerant nonpositional processor, functioning in a residual class system.

As individual components fail, the system of residual classes allows temporary loss of certain functions of a task in process. The probability of failure to perform any of the basic functions in a fixed time interval serves as a good measure of the "viability" of a processor.

Fault tolerance of processors is assured by introducing various forms of redundancy: Hardware, software and time.

Let us study an algorithm for implementing hardware redundancy in a system of residual classes. The digits of a nonpositional residue code are the least nonnegative residues of integers in mode $p_i(\forall_i \in [\overline{1,n}])$. The operation of taking the remainder (residue) $|X|_p^+$ is defined by the rule

$$\forall X \in Z : |X|_p^+ \rightleftarrows X - \left[\frac{X}{p_i}\right]^+ \cdot p_i, \qquad (1)$$

where Z is the set of integers.

The full system of representatives is the full system of least nonnegative residues in mode p_i and is represented by the symbol[4]

$$|\cdot|_{p_i}^{+} = \{0, 1, \ldots, p_i - 1\}.$$

Problems of coding numbers are reduced to the coding of elements in a ring of residues, while the matching of ring operations in a ring of residues with the arithmetic operations represents the arithmetic of finite models.

The advantage of nonpositional code is that it has the simplest structure with respect to circular operations of a ring $|\cdot|_p^{+}$.

If $p_1, p_2 \ldots p_n$ are pairs of mutually simple numbers, then $P_n = p_1 \cdot p_2 \cdot \ldots \cdot p_n$.

Suppose $X \pm Y \in |\cdot|_{p_n}^{+}$; then, based on the Chinese theorem of remainders assuming $|X|_{p_i}^{+} = \alpha_i$, $|Y|_{p_i}^{+} = \beta_i$, we obtain

$$\begin{aligned} |X+Y|_{p_i}^{+} &\leftrightarrow (|\alpha_1 + \beta_1|_{p_1}^{+},\ |\alpha_2+\beta_2|_{p_2}^{+}, \ldots, |\alpha_n+\beta_n|_{p_n}^{+}), \\ |X\cdot Y|_{p_i}^{+} &\leftrightarrow (|\alpha_1 \cdot \beta_1|_{p_1}^{+},\ |\alpha_2\cdot\beta_2|_{p_2}^{+}, \ldots, |\alpha_n\cdot\beta_n|_{p_n}^{+}). \end{aligned} \qquad (2) \qquad (2)$$

We can see from this that the circular (modular) operations in nonpositional code are parallel, assuring independent and simultaneous processing of all digits of the operands. Modular operations are the most important characteristics of nonpositional coding systems. In addition to modular operations, operations are also frequently performed which are positional in nature. As we know, a system of residual classes allows a significant (approximately ten times) increase in the speed of execution of such arithmetic operations as multiplication and addition. However, the implementation of nonmodular operations in a system of residual classes is quite difficult. The basic nonmodular operations are computation of positional characteristics and expansion of a system of bases, to which all other nonmodular operations can be reduced.

Devices implementing nonmodular operations are subdivided into two classes: Convolution devices and positional converters[5, 6].

Convolution devices represent a significant portion of the hardware of a nonpositional processor and are designed to convert numbers from positional systems of notation to systems of residual classes.

Conversion of a number to a system of residual classes can be performed by dividing the input number X in modulo p_i. The digits of the system of residual classes number are generated according to equation (1).

However, due to the division operation, the technical implementation of this method is not effective. A device was described in an earlier work for conversion of a number from a positional (decimal) system of notation to a system of residual classes which did not require the operation of division[7]. This device utilizes the following method to convert numbers: In any positional system of notation (including the decimal system), a number can be represented as

$$X = A_K N^K + \dots + A_1 N^1 + A_0 N^0, \quad 0 \leqslant A_i < N - 1, \quad 0 \leqslant i \leqslant K, \tag{3}$$

where N is the base of the system of notation.

Using the apparatus of comparison theory, the power of a base can be defined by the constant C_j:$C_0=1$, $C_1 \equiv N^1 \bmod p_j$, ..., $C_K \equiv N^K \cdot \bmod p_j$, and then

$$X \equiv (A_K C_K + \dots + A_1 C_1 + A_0 C_0) \bmod p_j = X_1. \tag{4}$$

The values of X_1 can be used to determine the remainder of division of X by p_j. If X_1 has more digits than p_j, the coefficient of number X_1 should be multiplied by the constants C_j, the sum produced $X_2 < X_1$.

This process can be continued until a number X_n is generated with a number of digits which is equal to or less than the number of digits of the modulus p_j, which will be a digit of the input number in the selected modulus in the system of residual classes.

Therefore, if $X_n = p_j$, the residue (remainder) is equal to zero, if $X_n < p_j$, the residue $\alpha_j = X_n$.

Thus, the input number is converted to another decimal number which is comparable to the initial number but has fewer digits, i.e., the length of the initial number is reduced.

A device converting a binary number to a system of residual classes differs from other known devices in that it requires the minimum hardware[8]. In accordance with Horner's equation, any binary number can be represented as

$$X \equiv (\dots((A_K \cdot 2 + A_{K-1}) 2 + \dots + A_1) 2 + A_0 \equiv \equiv X_1 \bmod p_j = \alpha_1 \bmod p_j, \tag{5}$$

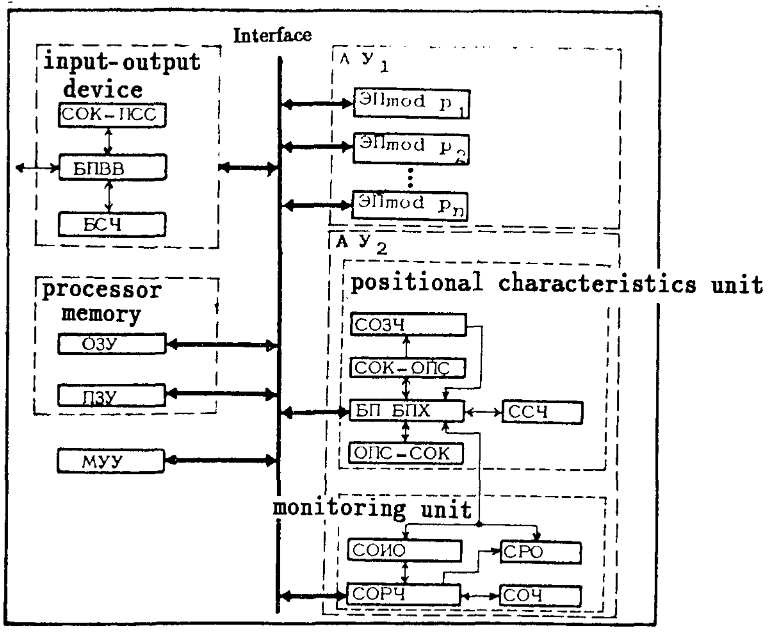
if

$$(A_K \cdot 2 + A_{K-1}) \cdot 2 \equiv X_K \bmod p_i;$$
$$(X_K \bmod p_j + A_{K-2}) \cdot 2 \equiv X_{K-1} \bmod p_j;$$
$$\vdots$$
$$X_2 \bmod p_j + A_0 = X_1 \bmod p_j = \alpha_j \bmod p_j.$$

As the device operates, the contents of the input register, except for the low-order bit A_0 and the high-order bit A_K, are shifted to the left. A modulo p_j adder is connected to the input of the high-order position. One peculiarity of this converter is that as the range of the numbers converted increases, the hardware costs increase only slightly. Thus, as the range is increased by 10^{10} times, the hardware costs increase by a factor of approximately four.

This device has been widely used among researchers studying the practical application of systems of residual classes, since it converts numbers very effectively.

Considering the short length of numbers, a nonpositional processor arithmetic device can be made as a set of tables to implement the basic modular operations.



The structure of a nonpositional processor is shown in the accompanying figure. The arithmetic device ($АУ_1$) can be made as individual paths equal in number to the number of bases, operating independently of each other and parallel in time, and formatted as normal elementary processors[5]. The need to perform nonpositional operations involving operations on an entire number requires that nonmodular processor $АУ_2$ be included in the arithmetic device, consisting of a positional characteristics module, containing a memory buffer (БП БПХ), plus a monitoring unit. The functions of the positional characteristics and monitoring units are: Determination of the sign of a number, comparison of numbers, division, rounding, determination of overflow, expansion of the system of bases, determination of the positional characteristics (rank, kernel) of the number and detection of errors. These functions must be performed to determine the position of numbers in the number range.

Methods related to determination of positional characteristics, not introducing ambiguity to their determination, can be reduced to conversion of numbers from systems of residual classes to a generalized positional system of notation[4]. Therefore, the positional characteristics unit includes a circuit to convert numbers from a system of residual classes to a positional system of notation (ОПС), the results of operation of which are used to determine the characteristics listed above. The positional characteristics can be determined sequentially with computation of the numbers in the positional system of notation. A system of residual classes-positional system of notation circuit is used to restore the numbers to the system of residual classes. Some position characteristics such as the sign of a number or the number of the interval in which a number is located utilize the values of the high-order bit of a number represented in a positional system of notation. This follows from the mutually unambiguous agreement between numbers represented in the system of residual classes and positional system of notation[4]. Therefore, the number sign determination circuit (СОЗЧ) is included in the positional characteristics unit.

An error in a code combination or overflow of the processor registers is also detected from the value of the high-order bit. This function is performed by the error detection and correction circuit (СОИО). Expansion of the system of bases requires knowledge of the bits in an unexpanded base system. This function is performed by the base expansion circuit (СРО), which is also contained in the nonmodular arithmetic device.

The functions of division, reduction of bases and rounding of numbers are performed by the number rounding circuit (СОЧ). Comparison of numbers or determination of intervals within which numbers are located requires knowledge of all digits of a number represented in a positional system of notation. This function is performed by the number comparison circuit (ССЧ), a part of the positional characteristics unit. The positional characteristics of rank and kernel of a number are determined by the number rank determination circuit (СОРЧ) on the basis of the values of the digits represented in the system of residual classes, and are used to detect and correct errors, expand the system of bases, for rounding, division and many other operations. The rank and kernel of a number make it relatively easy to implement nonmodular operations. As was shown in [9, 10], the rank and kernel can be calculated by means of modular operations.

A necessary element in a nonpositional processor, as in a positional processor, is the processor memory unit, consisting of random-access memory (ОЗУ [RAM]) and read-only memory (ПЗУ [ROM]); RAM is used to store numbers, ROM — to store programs and certain numerical constants.

The microprogram controller (МУУ) stores microprograms for all operations and organizes the operation of the entire processor by sending control signals to the basic hardware controller.

The input-output device (УВВ), consisting of a buffer input-output memory (БПВВ), and a unit for conversion of numbers from system of residual classes to positional system of notation and convolution of numbers (БСЧ), is included in a nonpositional processor for number input and output.

The devices and modules of the processor are connected by an interface system.

As we can see from the structural diagram of the nonpositional processor, the system of residual classes determines the specifics not only of the arithmetic device, but also of all other devices. It is basically reduced to subdivision of all processor hardware into individual independent channels, each of which corresponds to a given base of the system of residual classes, where the numbers, address portions of instructions and operation codes are all represented in the system of residual classes.

The structure of the nonpositional processor not only supports execution of the operations required, but also implements the major advantage of the system of residual classes with parallel execution of operations. All of this allows full implementation of parallel access to almost all processor units and allows development of a processor which does not have the shortcomings of classical computer structures.

The major peculiarity of the structure of a nonpositional processor from the standpoint of reliability is the potential for functioning of the processor after some of its units have failed. This allows operation to continue while reducing, within permissible limits, such quality characteristics as accuracy and speed. The capability of the processor structure to be tolerant of hardware and software faults allows the processor to operate in the gradual degradation mode. This property assures high viability of systems and is one of the major factors upon which the development of designs for fault-tolerant computer systems capable of gradual degradation is based.

Fault tolerance of a nonpositional processor is based on one of the most important properties of systems of residual classes — the capability for variation of accuracy, speed and reliability.

In contrast to a multiprocessor system, in this case the task is executed within a single nonpositional processor.

The structure of the nonpositional processor consists of independent channels which react in different ways to faults and failures of the hardware and allow the nature and structure of expected errors to be determined in hardware or software.

As an example of the functioning of a nonpositional processor which is resistant to failures in individual channels, let us analyze the computation of an integer polynomial in a system of residual classes.

Suppose the bases of the system are: $p_1=3$, $p_2=5$, $p_3=7$, $p_4=11$, $p_5=13$, $p_6=17$; of these, p_1-p_4 are data, p_5 and p_6 are test. The operating range $P=3\cdot5\cdot7\cdot11=1155$, the full range $P'=P\cdot p_5\cdot p_6=255255$. The orthogonal bases of the system $B_1=170170$, $B_2=51051$, $B_3=145860$, $B_4=46410$, $B_5=157080$, $B_6=195195$.

We must calculate the value of the integer polynomial where $x=10$:

$$f(x)=\frac{1}{2}x^4-3x^3-\frac{1}{4}x^2-\frac{15}{2}x-1851.$$

The polynomial is solved on a nonpositional processor with six independent channels corresponding to the selected bases of the system. The constants of the polynomial represented in the system of residual classes are in ROM, the values of the variable — in RAM.

Each action (raising to a power, conversion, multiplication) is performed in a single modular operation.

The final result is obtained in four modular operations

$$f(x)=\frac{1}{2}x^4-3x^3-\frac{1}{4}x^2-\frac{15}{2}x-1851=$$

$$=\left(2,\ 4,\ 1,\ 7,\ \frac{3}{2},\ \frac{12}{3}\right).$$

Computation of the value of the polynomial requires extraction of one number from RAM, four from ROM, plus eleven modular operations in the arithmetic unit.

If we assume that the RAM access time is equal to the time required to execute modular operations, computation of the polynomial requires 16 modular operations.

Let us now analyze the reaction of a nonpositional processor to faults and failures in a certain portion of the equipment. Let us assume that when variable x is retrieved from local memory of elementary processors ЭП mod p_1 and ЭП mod p_2, faults occur, for example, in two bits. After the fault in bases p_1 and p_2, variable x is multiplied by a number which contains zero digits in the corresponding positions, the product produced is therefore accurate and it is passed on for further computation, and with no further faults an accurate end result is obtained. If the bits in local memory of each elementary processor are represented in binary code, in this example faults can occur in five bits, and the result of the operation will still be correct.

Thus, the errors appearing as a result of faults are self corrected and normal functioning of the processor continues. Self correction of errors can occur not only with an individual fault, but also if several faults are superimposed in a given base. Self correction of errors eliminates second computation, doubling the effective throughput of the processor.

We note that checking of intermediate results is undesirable; only the final results should be tested.

The system of residual classes selected allows detection of double errors and correction of single errors. Errors may be either in the data or in the test bits. The final result is sent from the arithmetic unit through an interface into the buffer memory of the positional characteristics unit and then to the test unit. The value of the high-order bit of the number generated in the positional system of notation also flags errors, while the location of an erroneous bit is determined by the error detection and correction circuit by checking the correctness of projection of the number[11]. An individual error is corrected at this point.

We must note that, using the base expansion system, it is possible not only to detect, but also to correct errors in two tested bases.

Let us assume that during computation of the final result, two channels fail in test moduli p_5 and p_6. After location of the erroneous digits in the base expansion circuit, these errors are corrected. To correct the errors, zeros are written in the erroneous digits, then $f(x)=(2,\ 4,\ 1,\ 7,\ 0,\ 0)$. To restore the correct result, a method is used which is presented in work[5].

Even though two channels of the arithmetic unit fail during the process of executing the task (representing some 45% of the arithmetic unit hardware), the nonpositional process performs all functions with absolute reliability, though at lower speed — speed drops by a factor of 1.6. As individual elements or units fail, the throughput of the processor drops. If the number of operations involved in processing data is significantly greater than 10, the throughput of the processor does not decrease by much and the processor operates correctly when individual components fail.

The structure of a nonpositional processor can be designed to decrease degradation upon failure and assure more reliable processor operation, without requiring more complex software or introduction of additional hardware as is the case in traditional processors, while the correcting codes in a system of residual classes allow detection and correction of errors arising in all devices. Selection of the system of residual classes with the least base size increases the functional reliability of a nonpositional processor by relatively frequent self correction of errors, which is explained by the high probability of appearance of zeros in small base remainders. Furthermore, the throughput of the processor is retained with self correction.

REFERENCES

1. Prangishvili, N. V. Mikroprotsessory i lokalnye seti mikroEVM v raspredelennykh sistemakh upravleniya (Microprocessors and Local Microcomputer Networks In Distributed Control Systems). Moscow, Energoatomizdat Press, 1985, 572 pp.

2. Kagan, B. M. Elektronnye vychislitelnye mashiny i sistemy (Electronic Computers and Computer Systems). Moscow, Energoatomizdat Press, 1985, 552 pp.

3. Mamzeyev, I. A., Rusakov, M. Yu. "Fault-Tolerant Computer Systems," Zarubezh radioelektronika, 1983, No. 11, pp. 3-28.

4. Amerbayev, V. M. Teoreticheskiye osnovy mashinoy arifmetiki (Theoretical Principles of Machine Arithmetic). Alma-Ata, Nauka Press, 1976, 324 pp.

5. Akushskiy, I. Ya., Yuditskiy, D. I. Mashinnaya arifmetika v ostatochnykh klassakh (Machine Arithmetic In Residual Classes). Moscow, Sov. radio Press, 1968, 440 pp.

6. Dolgav, A. I. Diagnostika ustroystv, funktsioniruyushchikh v sisteme ostatochnykh klassov (Diagnosis of Devices Functioning In a System of Residual Classes). Moscow, Sov. radio Press, 1982, 65 pp.

7. Author's certificate number 374595, USSR, MKI^3 G 06 F 5/02. "Decimal Code To System of Residual Classes Code Converter," N. I. Chervyakov, Otkrytiya. Izobreteniya. 1973, No. 15, pp. 103.

8. Author's certificate number 374596, USSR, MKI^3 G 06 F 5/02. "Decimal Code To System of Residual Classes Code Converter," N. I. Chervyakov, ibid.

9. Klyaznik, V. V., Laskeyev, S. K. "Use of a System of Residual Classes In Design of Digital Filters," Vychisl. sredstva v tekhnike i sistemakh svyazi. 1978, No. 3, pp. 69-73.

10. Akushskiy, I. Ya., Burtsev, V. M, Pak, I. T. "New Positional Description of Non-positional Code and Its Application," Teoriya kodirovaniya i optimizatsii slozhnykh sistem (Coding Theory and Complex System Optimization). Alma-Ata, Nauka Press, 1977, pp. 8-16.

11. Author's certificate number 798846, USSR, MKI^3 G 06 F 5/02. "Device for Detecting Errors In Information Represented In a System of Residual Classes," N. I. Chervyakov, V. V. Lisunov, Otkrytiya. Izobreteniya. 1981, No. 3, pp. 208.

UDC 681.3

Verification of Circuit and Architecture Decisions

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian (manuscript received 23 Oct 86; after revision 14 Jan 87) No 3, May-Jun 1988, pp. 7-11

[Article by A. K. Kuzmin]

[Text] Errors in planning of integrated circuits require repetition of the expensive process of manufacturing these circuits. Improvements in solid-state technology and increasing levels of circuit integration have resulted in a significant increase in the cost of making changes and a tendency toward development of methods allowing detection of errors in the planning stage.

Modeling is most widely used to check circuit decisions. In the early stages of planning, modeling supports detection of errors, but in the later stages certain errors may remain undetected. The most serious problem is that modeling does not guarantee complete agreement between a circuit and its specifications; in other words, it does not guarantee full verification of a circuit decision.

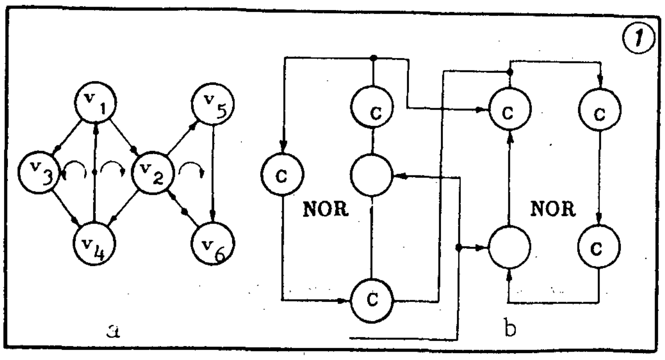
The creation of parallel systems has complicated the problem of verification. The number of states in such systems increases exponentially with the number of processors in comparison to a single-processor system. Furthermore, errors may occur in the planning stage which are characteristic of multiprocessor systems: Dead-end situations, from which the system cannot escape, incorrect system functioning. In general the sequence of execution of parallel processes is not rigidly fixed, i.e., the time at which a certain process starts relative to other processes may vary.

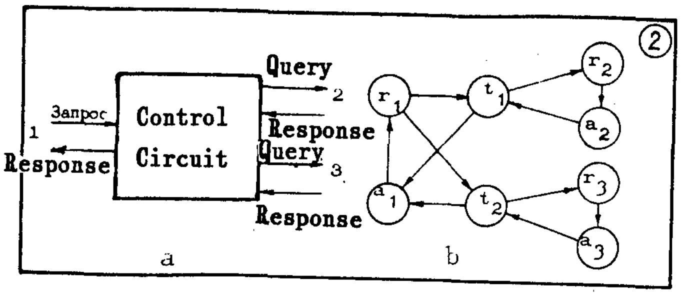
Relationships of partial ordering are more formally fixed in a set of parallel processes. A given task may be executed by different sequences of parallel processes[1-3]. This leads to additional difficulty in verification of circuit and architecture decisions for parallel systems.

A similar situation also arises in the creation of software. Full verification of programs by testing becomes practically impossible due to the tremendous number of states arising as the software runs. Program testing may quite effectively demonstrate the presence of errors, but unfortunately it cannot prove their absence[4]. This thesis has also become correct for hardware decisions as the degree of integration of elements and complexity of systems have increased.

The way out of this situation is to prove the correctness of circuit operation, in other words formal verification.

Programs reached a level of complexity at which testing could not provide an acceptable level of verification before hardware did. This explains the presence of a significant number of publications on program verification[5]. There have been fewer publications in the area of verification of circuit and architecture decisions.





However, the problem of verification of circuit and architecture decisions is now more acute than the problem of verification of software, due to the tremendous cost of correcting hardware errors. Software developers can permit the publication of incompletely verified programs, since they will be able to eliminate any errors detected quite rapidly. Hardware developers, who do not have this capability, are encountering the need for formal verification with increasing frequency.

Petrie networks. Computer hardware can be analyzed at several levels. At one level, a computer consists of simple memory devices and valves. At a higher level, the basic system components are functional units and registers. At a still higher level, entire computers may be components of computer networks. One of the major properties of Petrie networks is their ability to model each of these levels. However, in addition to modeling, we must also analyze the system modeled.

The most important tasks performed for Petrie networks are establishment of their safety and activity^6. Safety is a property guaranteeing correct functioning. The position of a Petrie network is safe if the number of connectors in it does not exceed a fixed integer *k*. Safety allows the position of the network to be implemented by a counter (any counter implemented in hardware is limited in terms of the maximum number it can contain). Activity of a network means that there are no dead ends. The task of analyzing Petrie networks is insolvable in the general case. Solutions have been found for certain subclasses of networks, and it is these networks which are most frequently used to verify hardware devices.

The problem of verifying circuit and architecture decisions based on Petrie networks has been the subject of publications[7-9].

Work [7] suggests a cellular structure for implementation of asynchronous control circuits. Cellular structures are usually not used to implement asynchronous circuits, since signal delays in cells make it more difficult to assure correct functioning.

The correctness of functioning is checked by means of a marked graph, in which sequences of circuit signals correspond to sequences of excitation of transitions. Marked graphs are a solvable subclass of Petrie networks. The correct functioning of a circuit is assured if its marked graph meets the conditions of activity and safety. A marked graph is active if each cycle contains at least one starting marker[6]. A marked graph is safe if each line contains no more than one marker in any state which can be reached from the initial state.

A safe marked graph can be implemented by replacing positions with Muller C circuits, initial markers with inverters. Figure 1 shows a marked graph (*a*) and its implementation (*b*). Work [7] describes a method of implementing a control circuit marked graph in a cellular structure.

Let us study a circuit for the control of parallel operations (Figure 2*a*). A pair of operations is started in response to the "query" signal on line *1*, which evokes the "query" signal on lines *2* and *3*, which actually start two operations. The completion of each operation is reported by the "response" signal. The "response" signals on lines *2* and *3* evoke the "response" signal on line *1*. We note that the second "query" signal may be fed to the control circuit before completion of operations performed in response to the first query. But operations are not restarted until they are completed, i.e., until the "response" signals are generated for the first start. The operation of this circuit is modeled by a marked graph (Figure 2*b*), which has the properties of activity and safety. Consequently, the hardware implementation of the graph by the method of [7] guarantees correct functioning of the control circuit.

This verification method is general in nature. First a Petrie network is generated modeling the operation of the circuit. The network must satisfy the conditions of activity and safety. Then the network is used to develop a circuit. Works[7-9] differ basically in the methods used to construct Petrie networks and implement them. Other applications of Petrie networks are discussed in review[10].

Predicate calculus. Yet another aspect of the verification of circuit and architecture decisions is related to the achievement of high computer reliability. In the opinion of specialists, the probability of failure of a computer used to control aircraft should not exceed 10^{-9} per 10 hours of flight. This is equivalent to a mean operating time between failures of about one million years, with routine maintenance each 10 flight hours. This extremely high circuit reliability generates serious problems with respect to reliability testing. Usually, system reliability verification is based on extrapolation of a certain range of observations. In the case of highly reliable systems, even observations of large numbers of specimens may be nonproductive.

Such a system retains its functionality with thousands of faulty elements. Consequently, verification by intentionally introducing defects is also unpromising. It was suggested in [11] that it be proven that a system satisfies the necessary requirements, even though this eliminates field testing.

A detailed description of a system allows more direct conditions to be used, but it is difficult to guarantee that a low-level description assigns the required properties.

A higher level description is required if we postulate the general properties of the system by a method which can be understood easily by man. This brings up the problem of transition from the high-level description to a detailed system description. In order for the high-level description to correspond to the actual system, we must formulate not a single description, but rather a hierarchy of descriptions. Each level in the hierarchy describes the properties of the system in terms of predicates and logical functions. Each level of description can be looked upon as an abstraction from the previous level. In order to test the agreement among the descriptions at various levels, it must be proven that any property of a higher level description is correct also for a lower level description.

The highest level of description represents the system requirements, with no information concerning the methods of their implementation. As we move downward through the levels of descriptions, additional mechanisms arise allowing incorporation of more detail concerning system operation.

In [11], the verification methodology is illustrated on the example of fault-tolerant computer systems implemented in software. Fault-tolerant computer systems are created by multiple duplication of processors, the results of computation are determined by voting. The difference between systems implemented in software and in hardware is that the voting, which is used to reveal and mask off processors which have failed, is implemented in software. The term "implemented in software" may create the impression that we are discussing verification of programs which contain errors, but this is not the case. The software facilities for guaranteeing fault tolerance require special hardware and are so closely interrelated that shortcomings in one of the components reduce system reliability. The methodology suggested for verification actually tests the sufficiency of both components.

A model of the highest level of system fault tolerance consists of axioms expressing limitations as to the order of iteration of various tasks and requirements for assurance of safety. The main axiom is formulated as follows: 1) In each iteration, task *a* is started and must be safe; 2) each task *b,* supporting an input of *a,* provides a single value in each iteration. In predicate calculus, the main axiom is as follows

$$a \text{ on during } i \wedge \text{task } a \text{ safe during } i \wedge$$
$$\wedge \forall\ b \in \text{inputs } (a) \mid \text{result } (b) \mid = 1,$$

where *a* on during *i* is a predicate which is true if task *a* is started during iteration *i*; task *a* safe during *i* is a predicate which is true if task *a* generates only one output result during iteration *i*; inputs (*a*) is the set of tasks preparing information for *a*; result (*b*) is the result of task *b*.

The second, more detailed level of description postulates the properties of the voting mechanism and the reconfiguration which occurs when a failed processor is detected.

At the third level of description, the properties of individual system processors are introduced. Each processor has a local clock, communications interface and buffers. Here also the asynchronism of the processors and its influence on their interaction must be described.

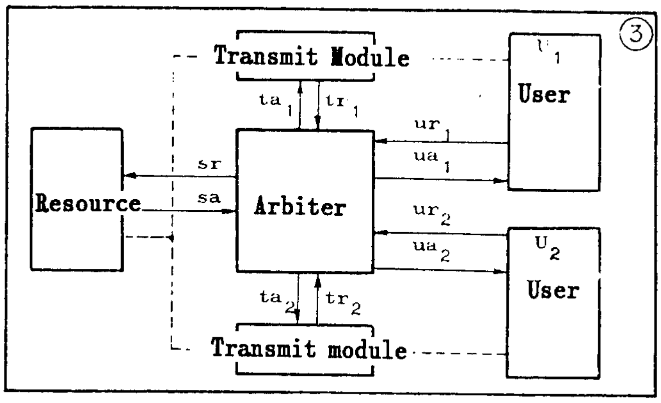
The fourth level describes the portion of the fault-tolerant system which is implemented in software in terms of pre- and post-conditions.

The correctness of the highest-level axioms must be proven at all description levels.

This methodology was used to verify the architectural decisions in a system consisting of eight BDX-930 processors. The formal verification process revealed planning errors which it would have been difficult, if not impossible, to detect in the testing stage. For example, it was proven that three independent clocks were insufficient for fault-tolerant synchronization. It was also discovered that transient faults could result in accumulation of computation errors.

It is interesting that verification of the second description level required some 22 proofs, verification of the fourth level required 17 proofs. It would have been virtually impossible to perform this task without automation of the proofs. The authors used a program for automatic construction of proofs in multivariate first-order logic.

Predicate calculus is also used to describe sequential circuits[12].



A λ-calculus is suggested in [13] for verification of circuits, which is also a variety of logical conclusion systems.

Temporal logic. One of the first attempts to use temporal logic for description of circuits was the development of vector-switching functions[14]. In addition to the ordinary boolean operations, vector-switching functions utilize operations on time intervals. These operations include delay, storage, and addition with storage.

A temporal logic is suggested in [15] which is an expansion of traditional logic and predicate calculus, for verification of hardware modules. It is shown that temporal logic is just as applicable to description of memory as to determination of the properties of safety and activity of processors represented by active circuits.

If statements in predicate calculus define the status of a system at a certain moment in time, called the current moment in time, temporal logic suggests that all possible sequences of states to which a system can evolve from the current state be defined. Let us present some typical temporal operators.

The "from this time" operator $\Box$ — the expression $\Box A$ means that statement A is true at the present time and throughout the future. For example, the statement $B \Rightarrow \Box A$ means: "Whenever B is true at a certain moment in time, A is also true and will remain true forever."

The "possibly" operator ∇ — the expression ∇A means that statement A will be true in a certain future, possibly at the present, but will not necessarily remain true. For example, the expression $B \Rightarrow \nabla A$ means that "if B is true, then A may become true." This operator is usually used to describe the possible response of a module to a certain query.

The "next" operator — the expression B next A means that if B is true at the current moment, A will be true at the next moment in time. This introduces the concept of discrete time. It is important to note that the use of the next operator is a result of idealization. It is assumed that transitions are shorter than the intervals assigned by the next operator.

The "while" operator — the expression A while B means that A is true at the current moment, since B is true at the current moment, and will remain true so long as B remains true.

The facilities of traditional logic are sufficient for description of combination circuits. Memory elements have two aspects — the setting of a new value and its preservation. These two aspects can be described partially by a certain function and the while statement.

If combination circuits and memory elements can be considered passive components, processes refer to those components of a system which cause changes in its status. The properties of processes are frequently classified in two ways: By safe statements and active statements. Safe statements describe properties which are performed at any moment in time (i.e., in variant statements) and properties determining the possible sequence of events.

Active statements describe properties which bring about certain methods of changing of states, corresponding to the desired behavior of a system.

The use of temporal logic is illustrated on the example of construction of an arbiter circuit, which defines the sequence of operation of two users with a shared resource (Figure 3). The arbiter interacts with the other modules by means of five pairs of circuits. The pairs (ur_1, ua_1) and (ur_2, ua_2) are used by modules U_1 and U_2 to request access to the resource. When the arbiter permits access by one of the users, information is transmitted between the user and the resource through the corresponding pair (tr_1, ta_1) or (tr_2, ta_2). When transmission is completed, the arbiter sends a request to the resource over the pair (sr, sa). After an answer is received, a new operating cycle can be started.

The sequence of appearance of signals in the circuits is as follows:

$$(r \equiv x) \Rightarrow (r \equiv x) \text{ while } (a \equiv \neg x)$$
$$a \Rightarrow \nabla \neg r.$$

The statement $\neg(tr_1 \wedge tr_2)$ describes mutually exclusive signals. The statement $ur_i \Rightarrow ur_i$ while $\neg sa$, $i=1, 2$ means that a transmission request remains active until a confirmation is received from the resource. The full system of statements describing the behavior of the arbiter will not be presented here.

Verification of circuit decisions consist of proof that the system of statements describing the arbiter can be derived from the statements describing the operation of the submodules of which the arbiter is made, considering the interconnections among submodules. In the proof process, the resultant statements are tested for activity and safety; testing for activity and safety is performed by introducing additional statements to the system describing the planned object and testing the derivation of these statements. For example, the statement assigning the property of activity is as follows

$$ur_i \Rightarrow \nabla(tr_i \wedge sr),\ i=1,\ 2.$$

This statement means that after a request of module i, the request may be serviced: Parameters are transmitted from module i and a request is made for the resource.

The use of temporal logic to verify circuits is also the subject of [12, 16].

Conclusions. Two trends can be distinguished in methods for formal verification of circuit and architecture decisions.

First trend — the planned circuit (architecture) is described by a model with known properties or the model is converted to a form satisfying certain limitations, which in turn provides known model properties. The most popular formal verification criteria are activity and safety of models developed. A circuit implementation of the models developed is then constructed. This trend includes planning with the use of various subclasses of Petrie networks (marked graphs, diographs, etc.).

Second trend — no limitations are placed on the construction of the model. The model is formulated in terms of predicate calculus or temporal logic. The properties of activity and safety are formulated as statements in the corresponding language. Formal verification consists of proof of the statements of safety and activity.

The first trend provides simple verification, but is not applicable to all systems.

The second trend may require significant effort to provide a proof, but is universal.

The problems of verification of software and hardware have a great deal in common. Attempts have frequently been made to transfer the methods developed for software to hardware, since software verification has deeper traditions.

The methods of formal verification developed to date are quite cumbersome. Their application is therefore more justified for hardware due to the great cost of errors in hardware.

The use of facilities for automating proof is growing in its significance.

REFERENCES

1. "Algorithms Software and Hardware of Parallel Computers." Edited by J. Miloshko, V. Kotov. Bratislava: VEDA, 1984, 395 pp.

2. Rabinovich, Z. L. "Machine Intelligence and Fifth Generation Computer Structures." Kibernetika, 1984, No. 3, pp. 95-107.

3. Seits, C. "Concurrent VLSI Architecture." IEEE. Translated on Comput. 1984, 33, No. 12, pp. 1247-1265.

4. Deykstra, E. Distsiplina programmirovaniya (Programming Discipline). Moscow, Mir Press, 1978, 275 pp.

5. Nepomnyashchiy, V. A. "Practical Program Verification Methods." Kibernetika, 1984, No. 2, pp. 21-28.

6. Peterson, J. Teoriya setey Petri i modelirovaniye sistem (Theory of Petrie Networks and System Modeling). Moscow, Mir Press, 1984, 263 pp.

7. Jump, J. "Asynchronous Control Arrays." IEEE. Translated on Comput. 1974, 23, No. 10, pp. 1020-1029.

8. Bunko, Ye. B., Yuditskiy, S. A. "Software Implementation of Petrie Networks In Asynchronous Logic Control Devices." Avtomatika i telemekhanika. 1983, No. 3, pp. 109-119.

9. Yuditskiy, S. A. Proyektirovaniye diskretnykh sistem avtomatiki (Planning of Discrete Automation Systems). Moscow, Mashinostroyeniye Press, 1980, 232 pp.

10. Nikonov, V. V., Podgurskiy, Yu. Ye. "Petrie Networks. Theory. Applications." Zarubezh. elektronika, 1984, No. 4, pp. 28-59.

11. Melliar-Smith, P. "Formal Specifications and Mechanical Verification of SIFT: Solt-Tolerant Fly Control System." IEEE. Translated on Comput. 1982, 31, No. 7, pp. 616-630.

12. Maruyama, F. "Hardware Verification." Computer, 1985, No. 2, pp. 22-32.

13. Gordon, M. "Register Transfer Systems and Their Behavior." Proc. Fifth International Conference Computer Hardware Description Languages. West Germany, 1981, pp. 23-36.

14. Rabinovich, Z. L. "Vector Switching Functions as a Language for Description of Information Processing Circuits and Processors." Kibernetika, 1968, No. 4, pp. 25-34.

15. Bohmann, G. "Hardware Specification With Temporal Logic." IEEE. Translated on Comput., 1982, 31, No. 3, pp. 223-231.

16. Moskowski, B. "A Temporal Logic for Multilevel Reasoning About Hardware." Computer, 1985, No. 2, pp. 10-19.

UDC 681.3.06

Control Facilities for an Automated High-Capacity Disk-Based Holographic Memory System for Text and Graphic Information

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian (manuscript received 23 Sep 86; after revision 06 Apr 87) No 3, May-Jun 1988, pp. 12-16

[Article by G. Sh. Vesna, I. N. Smirnov, B. G. Turukhano, A. Ya. Chernetsova and V. N. Yakutovich]

[Text] **Introduction.** The flow of scientific and technical information which can be represented as alphanumeric text, plans, drawings, etc. has significantly grown in recent years. This has greatly increased the requirements which must be placed on timeliness of processing, as well as information storage and retrieval structures and facilities.

One of the most promising answers to the problem of storing and reproducing text and graphic information is the use of holographic methods. The holographic method of recording text and graphic information provides high recording density, intelligent redundancy and high storage reliability with comparatively easy automation of all stages — from recording to reproduction. The holographic method of storage and reproduction of information allows rapid, noncontact reading, search for and page-level selection of information. It is also quite important that holographic methods allow recording, storage and reproduction of information as maps, drawings, diagrams, etc. without requiring any preliminary processing, as is required by memory devices using magnetic media[1, 2].

Thus, the holographic method of information storage is quite promising and its use can improve information processing.

Large capacity disk holographic memory system. Figure 1 shows an automated disk holographic system for recording, storage, retrieval and reproduction of information developed by the Leningrad Institute of Nuclear Physics, USSR Academy of Sciences, and introduced in a number of organizations[3].

The system includes the following elements: An information encoder, designed to create in computer memory a catalog of documents stored on the holographic disk; an automatic holographic disk recorder; an information machine, required to reproduce documents.

The basis of holographic memory is a matrix filled with Fourier holograms. It is a glass disk covered with a high-resolution photographic emulsion. Fourier holograms are located on the matrix in concentric circles. The capacity of the matrix is 10076 holograms 2.2 mm in diameter, each of which represents a page of text and graphic information.

Information to be stored on the holographic disk must have good contrast and be easily read under normal illumination. The information is transferred from paper to microfilm with a type AKM-22 camera and is sent on to the inspection equipment where its quality is checked and the documents are indexed (cf. Figure 1). Each imaged document is assigned a search pattern which is entered by the operator and stored in the holographic disk catalog, which is stored in the memory of an SM-4 computer. The search pattern is related to the nature of the information in the document and reflects its essence.

After collection, checking and classification, the microfilms are sent to the automatic holographic disk recorder. The positioning of the matrix, its marking, control of the location of holograms and movement of the matrix are controlled by an Elektronika-60 microcomputer.

Documents are reproduced in an information machine controlled by a computer (an Elektronika-60). Key parameters input by users in dialog mode are used to locate the required document (or group of documents) in the catalog. The coordinates of the document found are transmitted through a communications interface from the base computer to the control computer, which determines the required holographic disk position. An image of the document appears on the screen of the information machine. Modes provided include document "paging" (forward and backward) for multiple-page documents; examination of all documents with the same retrieval pattern; and transition to search for new key parameters.

Holographic memory system data. The large capacity disk holographic memory system has been tested at the Leningrad Institute of Nuclear Physics imeni B. P. Konstantinov (LINF) for the storage of nuclear data and patent information.

Nuclear data refers to information concerning the structure of nuclei and nuclear reactions, assembled in a nuclear data base currently containing over 200,000 documents.

Patent information consists of descriptions of authors' certificates and patents of the USSR, West Germany, England, France, the United States and Japan, plus annotations to patent descriptions in three areas: "C" — chemistry, "G" — physics and "H" — electricity.

Each document, both patent and nuclear information, is indexed, i.e., each document has an assigned retrieval pattern. The retrieval pattern of a patent information document contains the name of the country, international invention classification (IIC) and document number. The selection of the IIC as a retrieval key was based on the fact that it is assigned by all leading patent departments, and in most countries the IIC is the only classification used (or is supplementary to a national system). Thus, the IIC allows patent documents from various nations to be combined and uniformly systematized. The presence of the document number as a part of the retrieval pattern allows both thematic and numerical searches to be conducted.

The retrieval pattern of a nuclear information document includes (in coded form) the article title, last names of authors, the journal, its year of publication and other information of supporting bibliographic searches. In addition to the keys, the retrieval pattern of a document includes the coordinates of the document in the holographic disk archives and the number of pages of the document. The set of records corresponding to document retrieval patterns on a holographic disk makes up the catalog of the disk.

We present as an example the structure of a patent information catalog entry:

ISU G06F 1400 T00567172 001 001958 003

ISU is the patenting nation; G06F 1400 is the patent IIC (class, group, subgroup); T00567172 is the document number; 001 is the number of the holographic disk storing the document; 001958 is the number of the hologram on the disk; 003 is the number of pages of the document.

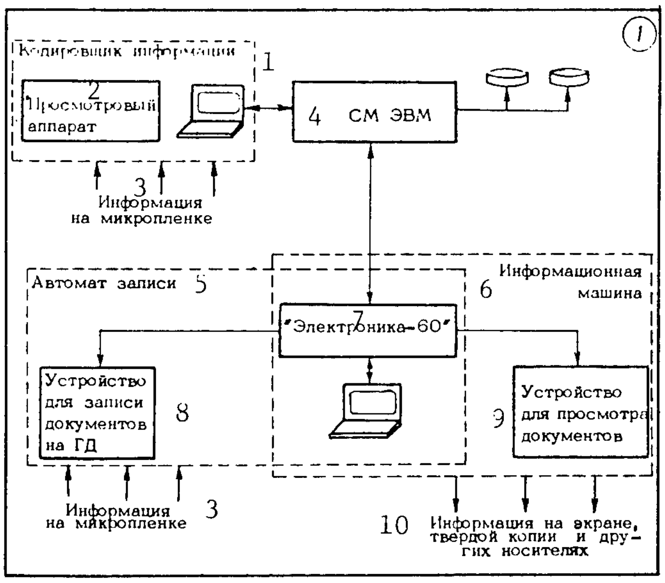
The data base of the automated holographic storage system was created using the KVANT-M data base management system (DBMS)[4]. This DBMS was selected for such features as independence of programs and data, the ability to maintain multiple files with multiple-key access, the variety of data description structures, field-level data access, etc.

The organization of the KVANT-M DBMS features files consisting of records of identical structure, each with a unique internal sequential number (ISN). Each record contains several fields (a field being the minimum named unit in a record), each of which has a unique name and an assigned format. In describing a file, any field in a record can be declared a descriptor, used as a key to access file records.

Basic Characteristics of a Record In the File NUCLEAR (Identifier Type ASC)

Data Abbreviation	Field Size	Field Descriptor Characteristic	Contents
DISKR	6	Y	Document number
NJUR	6	Y	Journal
NTOM	6	Y	Volume
NSTR	5	Y	Page
NGD	3	---	Disk number
NGOL	6	---	Hologram number
CLIST	2	---	Pages in document

The physical structure of each file in the data base is two areas: An associator and a data area. The data area contains all records of the file, including the ISN and data, compressed according to the field type. The associator contains inverted lists of descriptors and an address converter, providing independence of the address of a record from its actual physical location.


1
Кодировщик информации
2
Просмотровый аппарат
3
Информация на микропленке
4 СМ ЭВМ
Автомат записи 5
6 Информационная машина
7
"Электроника-60"
Устройство для записи документов на ГД
8
9
Устройство для просмотра документов
Информация на микропленке
3
10
Информация на экране, твердой копии и других носителях

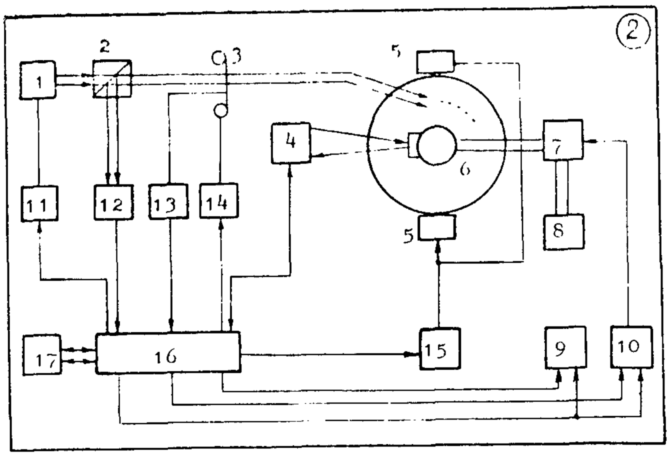
Key: 1, Information encoder; 2, reader; 3, information on microfilm; 4, SM computer; 5, writer; 6, information machine; 7, Elektronika-60; 8, holographic disk document recorder; 9, document reader; 10, information on screen, hard copy and other media.

The information base of this system is a set of independent files containing information catalogs stored on holographic disks: NUCLEAR — the nuclear data catalog; PATENT — the patent information catalog.

Schemas containing the field names, types and the data in the fields plus a characteristic indicating whether a field is a descriptor have been created to describe each file. Each file also corresponds to several subschemas which determine the location, type and number of fields necessary to perform a specific task.

In particular, the NUCLEAR file (table) contains seven simple fields, four of which are descriptors. Information is compressed as it is loaded into the data base: Trailing blanks are deleted from character information, leading zeros from numeric information.

The files of the holographic storage system can contain up to 2^{16} records and are stored on a single magnetic disk. Magnetic tapes are used for system backup and restore.



Disk holographic storage control hardware. The disk holographic storage system is controlled by a combination of hardware and software facilities. Stepping motors with ball-worm gears mounted on a frame make up the holographic disk head drive, used both in the automatic recording machine and in the information machine. This allowed the use of virtually identical holographic disk control hardware for both writing and reading of information. We shall therefore describe only the electronics controlling the process of writing on holographic disks.

The hologram write cycle in the automatic writing machine includes the following operations: Settling of the holographic disk, exposure, film advance and movement of the disk to record the next hologram.

Synchronous operation of the individual parts of the automatic recording machine in the proper sequence is supported by the electronic control system (Figure 2), based on Elektronika-60 computer *17,* which is coupled to the automatic recording machine through parallel input-output interface *16.*

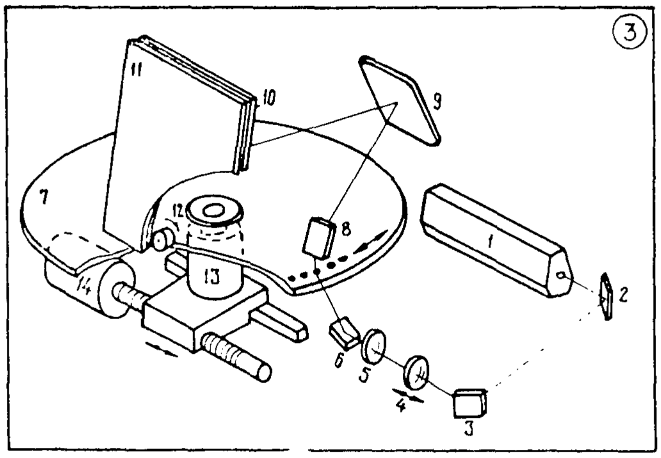
To allow settling of the disk, all supply voltages are disconnected from the optical-mechanical portion of the machine, and the electric brakes are released, fixing the disk in the desired position. Disconnecting the voltages decreases vibration at the moment when a hologram is recorded. The settling time is established by the operator controlling the recording operation, and monitored by the computer.

In order to assure good quality recording of a matrix using unstabilized laser *1*, an automatic exposure monitoring and adjustment system is used. The computer generates the signal which starts the laser *11*. A portion of the light flux from cubic prism *2* is then sent to a photoreceptor and analog-digital converter *12*. This digital information on the amplitude of each laser pulse is added by the computer. When the proper threshold is reached, exposure is stopped.

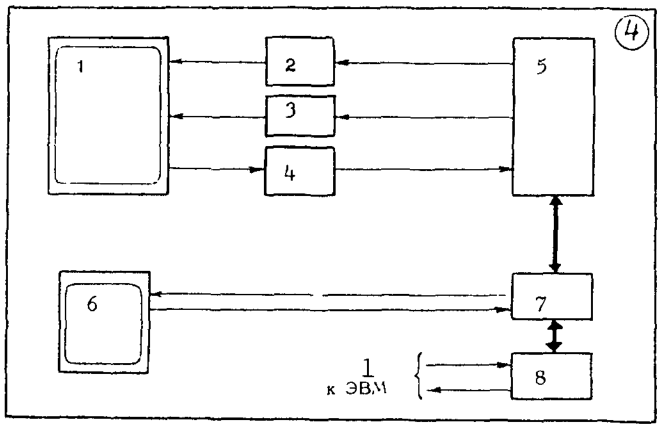
The film is advanced by film advance mechanism *3*, the electric motor of which is switched on by thyristor switch *14*. Should the film break, sensor *13* detects the break and stops recording.

The disk matrix *6* is advanced by the computer as follows. Thyristor switches *9, 10* receive the instruction to turn on the stepping motors. Following a delay equal to the time required for the stepping motors to begin operation, electric brakes *5* are disengaged through thyristor switch *15*. After this, the required number of pulses are fed from the computer to stepping motors *7, 8*, causing them to rotate. The voltages are switched off in the opposite sequence.

The initial position sensor *4* (light source, mirror on shaft and photodiode) allows the initial position of the disk to be set for writing and reading. An optical sensor is used to monitor the zero position of the disk, a mechanical sensor determines the radius.



Retrieval of documents, their reproduction on the screen for reading and selection are performed by means of an information machine, the optical-mechanical system of which is illustrated in Figure 3. Continuous gas laser *1* (LG-106 M) is used to restore the Fourier holograms from the matrix. The beam of this laser is collimated by two lenses *4* and *5* and directed by mirror *6* to matrix *7*. Mirrors *8* and *9* are used to reduce the dimensions of the reproduction device. The screen of the information machine is a decorative frame measuring 230×330 mm with two matte-finish scattering surfaces *10* and *11*. One matte surface is stationary, while the second rotates continually at 38 rpm (radius of rotation 1 mm). The disk is moved to select the desired hologram by two wave-type stepping motors *13, 14* in a polar system of coordinates. The use of the double scattering surface improves the scattering index, assuring more uniform distribution of screen brightness.

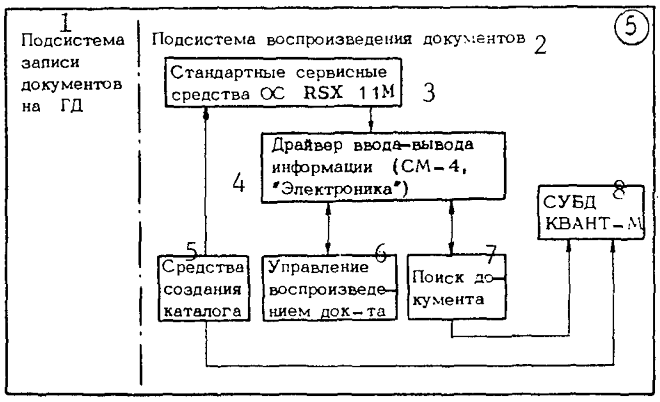


Key: 1. To computer.

The electronic control system of the information machine, diagrammed in Figure 4, contains a portion of the electronic recording control system. Also used are computer *7*, parallel interface *5*, display *6*, the power supply of the stepping motors *2*, *3* and initial position sensor *4*. However, while in the recording machine the voltage is completely disconnected from the stepping motors, in the information machine, due to the lack of electrical brakes, which decrease speed, the voltage is merely reduced to a certain level which causes the disk matrix to stop moving. This system also includes a standard IRPS radial serial communications interface *8*, which connects the information machine with the disk holographic storage system computer.

Technical Specifications of Disk Holographic Storage System For Text and Graphic Information

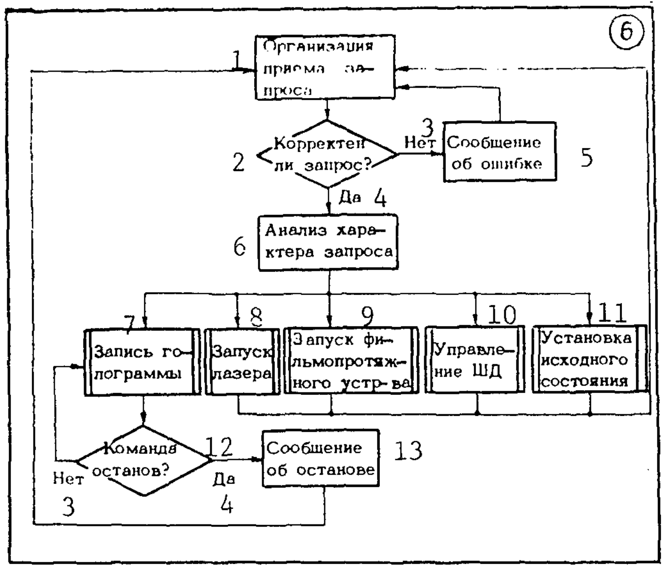
Disk matrix diameter	400 mm
Matrix capacity	10,076 pages
Hologram diameter	2.2 mm
Recording density	10^9 bits/cm
Recording speed	10^3 pages/hr
Mean time to prepare one disk of information	3 man-days
Input transparency size	24×36 mm
Mean document retrieval time including visualization	2 sec
Screen resolution	25 lines/mm
Speckle noise suppression in restored image	100
Recording radiation source (pulsed gas laser)	LGI-37
Reproduction coherent radiation source (continuous gas laser)	LG-106 M
System operating modes	Autonomous, on line with computer



5
1 Подсистема записи документов на ГД
Подсистема воспроизведения документов 2
Стандартные сервисные средства ОС RSX 11М 3
Драйвер ввода-вывода информации (СМ-4, "Электроника") 4
СУБД КВАНТ-М 8
Средства создания каталога 5
Управление воспроизведением док-та 6
Поиск документа 7

Key: 1, Holographic disk storage subsystem; 2, document reproduction subsystem; 3, standard RSX 11M operating system service facilities; 4, data input-output driver (SM-4, Elektronika); 5, catalog generation; 6, document reproduction control; 7, document retrieval; 8, KVANT-M DBMS.

Automated holographic storage system software. The disk holographic storage system control software was developed using an SM-4 computer in the RSX environment and an Elektronika-60 microcomputer. Figure 5 shows the general structure of the software, which consists of two independent subsystems: Holographic disk document storage and reproduction control.



Key: 1, Receive request; 2, request correct?; 3, no; 4, yes; 5, error message; 6, analyze request; 7, write hologram; 8, start laser; 9, start film transport; 10, control stepping motors; 11, set initial status; 12, stop instruction?; 13, stop message.

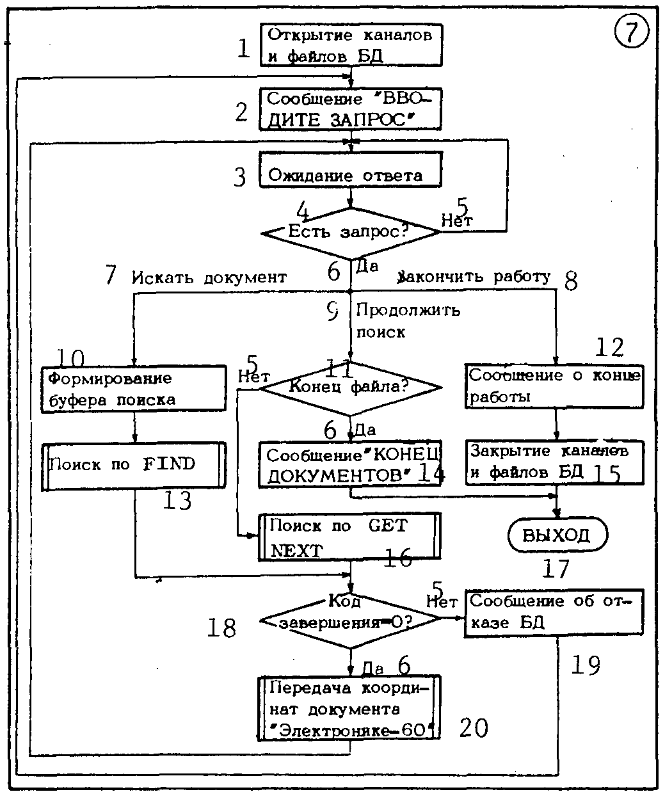
The document storage subsystem software (Figure 6), which controls the operation of the recording machine, performs the following major functions:

— Set initial matrix status;
— assign hologram number;
— start and stop electromechanical devices;

— write hologram on matrix;
— shut down automatic recording machine.

In addition to these functions, the software can set and change the integral exposure threshold, stepping motor settling time and stepping motor speed, both angle and radius.

When the subsystem is started, all programs involved in the cycle of writing one hologram are initialized in sequence. Upon completion of the cycle, the hologram counter is incremented and the cycle is repeated until the entire matrix is filled or a stop instruction is generated (planned completion of operation, laser fault, etc.).


7
1 Открытие каналов и файлов БД
2 Сообщение "ВВОДИТЕ ЗАПРОС"
3 Ожидание ответа
4 Есть запрос?
5 Нет
6 Да
7 Искать документ
8 Закончить работу
9 Продолжить поиск
10 Формирование буфера поиска
11 Конец файла?
12 Сообщение о конце работы
13 Поиск по FIND
14 Сообщение "КОНЕЦ ДОКУМЕНТОВ"
15 Закрытие каналов и файлов БД
16 Поиск по GET NEXT
17 ВЫХОД
18 Код завершения=0?
19 Сообщение об отказе БД
20 Передача координат документа "Электронике-60"

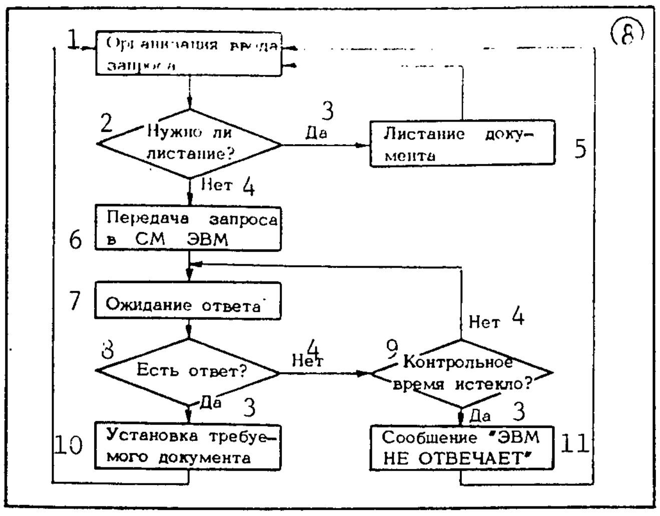
Key: 1, Open channels and data base files; 2, "ENTER REQUEST" message; 3, wait for response; 4, request made?; 5, no; 6, yes; 7, locate document; 8, stop operation; 9, continue search; 10, generate retrieve buffer; 11, end of file?; 12, operation complete message; 13, search using FIND; 14, "END OF DOCUMENTS" message; 15, close channels and data base file; 16, search using GET NEXT; 17, end; 18, completion code=0?; 19, data base error message; 20, send document coordinates to Elektronika-60.

The parameters necessary for operation of the subsystem are input by the operator from his terminal in dialog mode. The write subsystem is implemented on an Elektronika-60 computer, the programs were written in Assembler.

The read subsystem includes the software necessary to create catalogs, plus facilities for retrieval, identification and visualization of documents.

The lines of the catalog, containing the retrieval pattern and coordinates of documents on the holographic disk, are input by the operator at the terminal. The text editor EDI or screen editor TED is used to generate the initial catalog file, which is then entered into the data base by the LOADER program in accordance with the file description, loading schema and subschema. Utilities of the KVANT-M DBMS are used to monitor the information placed in the data base, protect it and enter new records in the catalog.

To select documents from the holographic disk archives, the document retrieval program, diagrammed in Figure 7, is run. A user request input at the information machine terminal contains one or more descriptors of the desired document. The KVANT-M DBMS can search in logical or random descriptor sequence. When using logical sequence, access to records is performed in the sequence of increasing values of the descriptors selected. The descriptor value indicated in the request is used as the initial point for successive examination by inverted lists. The physical address of the records found is constructed by the address converter. Random access is used in response to a request consisting of a combination of several keys indicating the value or range of values of several descriptors. The result of processing such a request is a logical combination of several ISN series.


(8)
1 Организация ввода запроса
2 Нужно ли листание?
3 Да
5 Листание документа
Нет 4
6 Передача запроса в СМ ЭВМ
7 Ожидание ответа
8 Есть ответ?
4 Нет
9 Контрольное время истекло?
Нет 4
Да 3
10 Установка требуемого документа
Да 3
Сообщение "ЭВМ НЕ ОТВЕЧАЕТ" 11

Key: 1, Input request; 2, paging required?; 3, yes; 4, no; 5, document paging; 6, send request to SM computers; 7, await response; 8, response present?; 9, time out?; 10, determine required document; 11, "computer not responding" message.

When a document is located, its coordinates are sent to the Elektronika-60 computer which controls the information machine. A flow chart of the program determining the required matrix position is shown in Figure 8.

Information is transmitted over the base system computer to the control computer by means of a special input-output driver. In addition to determining the required document, this program also "pages" forward and backward if a document contains more than one page.

This disk holographic storage system is a passive system, since only information which has been stored in it can be retrieved. However, as operating experience has demonstrated, if a sufficiently large number of documents is stored in the system, it can perform rapid retrieval, which is practically impossible in a data base of over 10^6 pages, and also automatically select documents based on their keys (for example, select materials for composition of reports, reviews on various themes, etc.), leading to a significant savings in working time.

This system is highly noise tolerant, insensitive to defocusing, and invariant to shifting of holograms relative to the reading laser beam.

REFERENCES

1. Tolchin, V. G., Turukhano, B. G. "Disk Holographic Storage System." Materialy VI vsesoyuznoy shkoly po golografii (Materials of Sixth National Holography School). Leningrad, Leningrad Institute of Nuclear Physics, USSR Academy of Sciences, 1974, pp. 303-328

2. Turukhano, B. G. "Holographic Aspects of Memory." Fizicheskiye osnovy golografii (Physical Principals of Holography). Leningrad, Leningrad Institute of Nuclear Physics, USSR Academy of Sciences, 1978, pp. 56-69

3. Turukhano, B. G. "A Disk Holographic Storage System." Opticheskaya golografiya prakticheskiye primeneniya (Optical Holography. Practical Applications). Leningrad, Nauka Press, 1985, pp. 75-95

4. Opisaniye sistemy upravleniya bazami dannykh EVM linii SM-4 KVANT M (Description of the KVANT-M Data Base Management System For SM-4 Computers). Moscow, Institute of Atomic Energy imeni I. V. Kurchatov, 1982, 204 pp.

UDC 681.3.06

The YaK-3 Electronic Circuit Check Language

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian (manuscript received 08 Dec 86) No 3, May-Jun 1988, pp. 17-21

[Article by A. S. Dolgopolov and Ye. M. Zamoruyeva]

[Text] **Introduction.** A system description language [1, 2] is the input language used to create automatic (or automated) electronic equipment test generation systems based on hardware descriptions. Such systems cannot yet satisfy all the requirements of practice, since their capabilities have lagged behind the increasing complexity of electronic equipment. This limits the area of their application to relatively simple modules.

This has resulted in the appearance and utilization of task-oriented check languages, which are algorithmic languages, in contrast to circuit description languages, for the development of test programs for electronic equipment modules and systems. These languages can describe the test algorithm for a module of virtually any complexity, but the composition of an algorithm requires a deep understanding of the laws governing the functioning of the tested object and, consequently, the designer of the unit itself must be involved in the task.

The number of check languages in existence today [3, 4] is sufficient to threaten a "tower of babel" situation in this area. More than a quarter century of history of the development of programming languages indicates the difficulty of solving such problems. The element of subjectivism in evaluating language system quality leads to their "natural selection" based on the experience of using a language. Unfortunately, the experience of using check languages is still limited (much less than the experience of using traditional programming languages). This makes well-founded decisions more difficult. This difficulty can be partially overcome by describing the practical work done on the development of check languages, with discussions of decisions made and results achieved. As a contribution to this effort, we describe here the YaK-3 check language, its implementation and our experience in its application.

Peculiarities of check languages. The prerequisites for the application of many check languages result from the differences which exist in the conditions under which actual testing processes occur (differences in classes of objects tested, composition of testing and diagnostic equipment, test goals, etc.). However, there are but two basic situations in which differences in approaches to the design of check languages are significant.

In one, there is a "doctor-patient" relationship between the test system and the tested object, i.e. the test system is independent of the tested object, but can act upon it and analyze its reactions, whereas actions in the reverse direction are minimal. This is the situation which always obtains if neither the tested object nor the entire system has facilities to implement an assigned testing algorithm.

In the second situation, the tested object is (or is artificially made to be) a structural element of a certain intelligent system and is tested by some form of self-testing ("self-diagnosis") system. In this case, the tests are created by professional programmers using the linguistic and operational facilities of the system, and the problem of a special check language does not arise, or is solved by traditional programming methods — libraries of macros, subroutines, etc. (this situation usually arises in testing of high-level units — system instruments and devices).

Let us analyze the first of these situations and utilize the term "check language" as applicable to it. We note that this is the case in which it is possible to test and diagnose low-level units under series production conditions.

Since check languages are algorithmic, it is desirable to use the experience gained in the creation of ordinary programming languages during their development, conserving the traditions of these efforts:

— Operator structure of program text, with distinction among description, control, assignment and input-output operators;

— the concepts of the identifier, file, index, label and decimal constant, together with the syntactic elements traditionally used (for example, name and label — alphanumeric lexemes beginning with letters, the ":" as a distinguishing characteristic indicating a label, etc.);

— the syntax of an assignment operator A=B (or A:=B) with the functional-operational structure of the write portion B, etc.

Check languages are designed for a broad range of users, particularly electronic equipment design engineers. This demands of such a language maximum simplicity and intelligibility of syntax and semantics, orientation to the terminology and procedures utilized in testing processes. The requirements of clarity and "readability" of programs should also be met, since programs are used as technical documents describing the methods used to check products. In particular, the structure of the language should be as close as possible to natural language statements. The language should be as brief as possible, since brevity significantly reduces the labor involved in writing and using programs.

It is therefore not desirable to include in the language terms which are difficult for nonspecialists to understand and master including, in the opinion of the authors, such things as:

— Program structuring facilities (blocks, procedures, the concepts of local and global variables, etc.);

— complex control functions (switches such as the assigned GO TO, complex loops, etc.);

— complex data types and input-output specifications.

In designing a check language, one must also recall that the test system instruments include mini- and microcomputers. The designer should therefore create effective, fast systems for translation and interpretation using limited resources, i.e., the design of the language should be approached from the standpoint of minimizing its imaging facilities. It is the optimal selection of linguistic facilities based on these criteria (in addition to the creation of practically effective systems) which, in our opinion, is the most pressing problem of today. We should note that there is a tendency to replace this with theoretical development of "superhigh level" languages with the maximum capabilities, along with the maximum problems of their implementation[4].

The operating environment of check languages includes data of a special type, reflecting the physical signals which are exchanged by the test system and the tested object. For example, for digital objects these include binary vectors of significant dimensionality (up to several hundred), with complex internal structure. Accordingly, the language must include facilities for processing such data, and practice has shown that the convenience and flexibility of these facilities have a great influence on linguistic quality in general. In particular, they cannot be treated as simple masses of bits, as, for example, in PL–1.

One concept for working with such data was implemented in the TEST language[3] (and borrowed in a form expanded for testing of digital objects in the OKA language[4]). In the opinion of the authors, it is insufficiently flexible. In particular, structural elements are identified by ordinal numbers (indexes) of the component bits or by means of another bit array (mask) of equal dimensionality. This leads to rather cumbersome expressions, a loss of program readability, and significant difficulties in programming complex structures. Effective machine implementation of operations on elements of such structures is quite difficult.

The following approach is used in YaK–3. Elements of a binary structure are considered ordinary integer–type variables, processed by an ordinary assignment operator, but with an expanded operation set, and their combination into structures, including dynamic combination, is performed by special operators.

Operators (which we call signal operators) converting information objects into physical signals and vice versa represent a special group in check languages. From the standpoint of system programming they are input–output operators. However, it would not be correct to interpret them as such in a check language. They must rather naturally reflect testing procedures from the standpoint of the user. The degree of development of this group of facilities and their level of task orientation also represent one of the most important components of a check language.

Description of the YaK–3 language. For purposes of brevity and clarity, we will base our description not on the Baccus metalanguage, but rather on specific examples, turning particular attention to operator semantics.

The alphabet of the language includes the characters of KOI–7 code. The identifiers (names) of variables, arrays and labels are alphanumeric lexemes of arbitrary length (only the first six characters are used). The "period" and "underline" are permitted within names — they are considered letters in this case. This allows the use of such names as OUT. VOLTAGE, INPUT SYNCHRONIZATION, etc., along with their abbreviated versions.

There are two basic data types: INTEGER and ANALOG (corresponding to real numbers). There are five types of constants: Decimal integer, octal, binary, hexadecimal, floating point, for example: •101010 — binary, ′177776 — octal, ⊛ AFFF — hexadecimal.

One–dimensional arrays of variables and indexed constants are allowed (the language may be expanded here, though the need has not been demonstrated in practice).

A subclass of "Standard names" is included, which do not require description in user programs. They are designed to orient the language to specific hardware devices and classes of tested objects. Standard names represent test system instruments, their ranges and modes, test system channels, etc. From the standpoint of implementation they are built into the language translators. When necessary, the user can specifically represent nonstandard names by means of the DESIGNATE operator:

DESIGNATE FREQUENCY/K17, STROBE/K9, ...,

where K9 and K17 are standard names (for example, of digital programmable test system channels), FREQUENCY, STROBE are user names related to the tested object.

The assignment operator has the traditional form: A=B, where A is a simple or indexed variable, B is an arithmetic–logic expression. However, the list of operations in B is significantly expanded, and furthermore is left open.

Two–position shift operators of various types and a number of special operations have been included: EVEN, ODD, AND, NAND, etc., for example: A=B<<-2 (low–order byte of A assigned value of low–order byte of B, cyclically shifted left two bits); A=B->K(M) (word A equal to word B, logically shifted to the right by K(M) bits); A(K)=A(K)&'1010 (element K of A logically multiplied by the binary number 1010).

Abbreviated recording of an operator is permitted when the left portion (A) coincides with the left operand of the right portion (B). For example, the last operator is equivalent to: A(K) &'1010.

Practice has shown that this solution yields a significant reduction in check program size, in which structures such as ADDRESS+1 (increment address by 1), ¬ CHANNEL (bitwise in version of channel) are frequent.

Special operations perform actions typical in testing tasks. For example, suppose the tested object has a two–byte data width, and each byte has a parity check bit. A signal is generated on the bus by representing it in four variables: BYTE1, BYTE2, CB1, CB2, where BYTE1 and BYTE2 can be assigned arbitrarily, while CB1, CB2 are calculated as CB1=EVEN BYTE1; CB2=EVEN BYTE2 (the operation EVEN computes the mod 2 sum of ones in an operand). For greater clarity, operands are not placed in parentheses unless required for unambiguity.

Combining of variables into a common binary structure is performed by two main operators. The first provides a static description of a group of variables:

<name>#<name list>. (1)

The semantics are as follows: The variable <name> is a binary sequence which is the concatenation of the binary sequences in <name list>. For example:

BYTE#K1, K2, K3, K4, K5, K6, K7, K8;
CB1#K17.

Here K1, K2, ... are the standard names of digital programmable test system channels, i.e., the operators describe the connections of the test object with the test system and are naturally understood by the user.

The second of these operators is dynamic and assigns descriptions to variables which have been declared by the first operator:

<name 1>==<name 2>.

The semantics are as follows: Variable <name 1> becomes a structured variable with <name list>, in the operator <name 2> #<name list>. This operator is particularly useful for testing objects which have several similar circuits (for example, channels). In this case, it works as a software switch, connecting the physically different channels of the object tested to the same test program without changing the electrical switching.

The machine implementation is as follows. A digital programmed signal buffer is reserved in the system — a binary file of sufficient dimensions. Each bit corresponds to a certain standard (for example, K1, K2, ...). The description operator (1) is translated to special tables which are stored together with the object program. The assignment operators, in addition to their usual action, access these tables and, if the variable in the left portion is included in the structure, fix the result of the assignment in the buffer. Transmission of the buffer directly to the tested object or comparison of the reaction of the tested object with its elements is performed by the signal operators (such as SEND, TEST), which are studied below. Feedback from the tested object is produced by an assignment operator with the special operation MEASURE, for example:

```
                    CODE#K16, K4, K7

                   . . . . . . . . . . .

                  CODE2=MEASURE CODE
```

assigns the variable CODE2 to the set of three digital signals CODE directly from the tested object, connected to channels K16, K4 and K7. This provides extensive capabilities for automatic diagnosis of defects in the tested object.

Control operators in YaK-3 are quite traditional and we shall limit ourselves here to a few examples: GO LABEL — unconditional transfer to a label; IF A$\rangle$=16 GO LABEL — conditional transfer if A$\geqslant$16; REPEAT 16, LABEL — loop operator, repeated 16 times (the previous sequence of operators, beginning with LABEL).

The operators also include SUBROUTINE, RETURN (from subroutine), END (stops testing process and indicates results, PASS-FAIL), WAIT (synchronizes the program with events in real time). To give these operators more natural appearance, comments can be made in certain parts of the operator bodies (comments in bold):

IF REJECT **THEN GO TO TEST NAND CIRCUIT TO A LABEL** M3
REPEAT K **TIMES BEGINNING AT LABEL** M3.

The basic semantic set of the signal operator group is quite similar in many check languages. In YaK-3 it consists of the operators: CONNECT — controls automatic switching of the tested object with the testing system, SET — sets operating modes of test system instruments, SEND — initiates transmission of input actions to tested object, STOP — disconnects input actions from tested object, CHECK, COMPARE — checks the reaction of the tested object to the input actions, MEASURE — measures the reaction of the object and converts it to digital form.

For example:

```
CONNECT FREQUENCY — GCh1,
VOLT.PWR — IPT2,
OUT.VOLT — VTs1
```

means connect (automatic) points FREQUENCY, VOLT.PWR and OUT.VOLT of the tested object to the input-output devices of the test system — frequency generator No. 1 GCh1, dc power source No. 2 IPT2 and digital volt meter No. 1 VTs1. Here GCh1, IPT2 and VTs1 are standard names;

```
SEND FREQUENCY = 10 kHz,
VOLT.PWR = -24 V
MEASURE OUT.VOLT = VTs1
```

initiates input actions, measures the signal at the output of the tested object in digital form with the instrument VTs1, placing the result in the variable OUT.VOLT.

This set is insufficiently flexible for testing and diagnosis of digital objects, since there are many standard actions related to testing of objects of this type which can be represented more naturally and effectively by specialized operators. For example, the ordinary method of switching digital tested objects is rigid connection with programmable test system channels, and testing requires merely the assignment of the distribution of test object inputs and outputs among channels. This switching is most conveniently reflected by the operators INPUTS <name list>, OUTPUTS <name list>, where <name list> is a list of the names (standard or introduced by operator [1]) of the input and output signals, respectively. The INPUTS and OUTPUTS operators in YaK-3 are dynamic, allowing switching of the tested object during the course of a test (for example, to test objects with bidirectional lines).

Frequently when digital objects are tested, a fixed sequence of actions in the basic semantic set is used: SEND — MEASURE — COMPARE. In YaK-3, this corresponds to a single operator TEST <specification of input actions and standard reactions>. To prevent testing of object reactions which take on uncertain or unknown values for a certain time interval, the dynamic MASK operator is used: MASK <name list>, where <name list> lists the signals which are to be tested in the TEST, CHECK and COMPARE operators.

Modern test systems frequently include hardware channels which implement the most common interfaces. The operators READ and WRITE are used to exchange actions and reactions through such channels. For example:

```
WRITE COM.BUS,
ADDRESS = '1000, WORD = 0
```

means write a zero work through the "common bus" channel at address 1000_8;

```
READ COM.BUS,
ADDRESS = ADR, B = BYTE
```

means read a byte from the address in variable ADR into variable B. Here COM.BUS, ADDRESS, WORD and BYTE are standard names. Data arrays of a standard form ("checkerboard" type) are exchanged in a similar manner, greatly reducing the time required to test such objects as memory units.

Test results are indicated by the operators PRINT and PRNT.FAIL (another peculiarity of a check language). In test processes there is always an unstated logical variable (in YaK-3 it consists of the standard names PASS-FAIL), and in the initial state has the value PASS, taking on the value FAIL if a deviation from normal functioning of the tested object is found to end any test. The difference between the operators PRINT and PRNT.FAIL is that the former does not influence the PASS-FAIL variable, while PRNT.FAIL sets it to FAIL (a similar relationship exists in the pair of operators CHECK-TEST). Upon completion of the testing process, the variable PASS-FAIL is analyzed by the END operator and the corresponding indicator is sent to the monitoring post.

Output is formatted by the message text element description operator TEXT: TEXT *n*/<text constant>, where *n* is an integer, identifying the <text constant> which follows it. An output example:

```
TEXT 1/DEFECT.LINE 2, CODE =
      TEXT 2/, LOOP
 . . . . . . . . . . .
IF FAIL THEN PRNT.FAIL 1/CODE,
         2/LOOP.
```

The result will be the message:

```
DEFECT.LINE 2, CODE = '77, LOOP 4
```

(here '77 and 4 are the current values of the variables CODE and LOOP).

Implementation and usage experience. The YaK-3 language has been used for several years as an input language of three types of stands for testing of digital modules. The base machine of the test stands is an Elektronika-60 computer. The software is based on the YaK-3 actuating system, written in macroassembler for the FODOS operating environment. The actuation system includes a kernel containing a translator, an interpreter for the hardware-independent elements of the language, dialog program debugging system and system for communications with the test-stand operator. The kernel of the system is implemented together with the driver of a specific test system which implements the hardware-dependent elements of the language and an adjustment module. This module is used to set the standard names, distribute memory and perform other actions involved in generating a specific actuation system version.

The translator translates programs from YaK-3 to an internal intermediate language. The decisions made in designing the language, discussed above, have allowed completely resident operation of the actuation system along with programs in the intermediate language with lengths of up to 2-3 thousand operators in 28 Kwords of RAM. The intermediate language retains all information on the initial program necessary for source-language debugging.

The debugging facilities contain a full set of functions used in such systems — dialog control of program execution, indication, tracing, etc. We note that the output of the debugging system (for example during tracing, single-step execution, etc.) is also represented in the source language by partial reverse translation of the intermediate language. The very high translation speed (on the order of 3000 operations per minute) does not require storage of object modules in the intermediate language in internal memory (on floppy disks). In addition to the savings of disk storage space (which is rather sparse in floppy-disk systems), this significantly improves the ease of use of the system (in particular, the user does not need to learn the details of file system organization on the floppy disks, the frequency of data transfer to and from the disk is significantly reduced, etc.).

In two years, more than 50 testing programs for complex digital modules (containing over 100 MSI and LSI chips, including microprocessors and memory elements) have been written in YaK-3. Most of these programs are now in commercial use. The mean length of a program is 500-700 statements, the maximum — up to 1.5 thousand. Test programs are developed and debugged by the hardware designers themselves. Although most of them are first-time software product producers, the YaK-3 language, our experience has shown, is easily mastered by this range of users and can be effectively used to perform testing tasks.

REFERENCES

1. "Commercial Test Planning Automation System." O. F. Nemolochnov, A. Ye. Usvyatskiy, V. F. Zvyagin and V. N. Golynichev. USiM, 1981, No. 5, pp. 37-42.

2. "Automated Test Design System For Highly Integrated Digital Circuits." A. G. Birger, A. V. Boyarshinov, M. Yu. Vinokur, Ye. S. Ryzhkov. Obmen opytom v radiopromyshlennosti. 1983, No. 4, pp. 4-9.

3. Problemno-oriyentirovannyy yazyk programmirovaniya TEST (The TEST Application-Oriented Programming Language). A. Ye. Podzin, K. Sh. Ibragimov, I. Kh. Kornya, B. Z. Kirilenko. Kishinev, Shtiintsa Press, 1978, 127 pp.

4. Proyektirovaniye vneshnikh sredstv avtomatizirovannogo kontrolya radioelektronnogo oborudovaniya (Planning External Automated Test Facilities For Electronic Equipment). Edited by N. N. Ponomarev. Moscow, Radio i svyaz' Press, 1984, 283 pp.

UDC 681.32

Properties of Programmable Logic Matrix and Logic Function Decomposition

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian (manuscript received 09 Jul 86; after revision 09 Nov 87) No 3, May-Jun 1988, pp. 21-23

[Article by V. I. Lazarev]

[Text] The task of decomposing a logic function in order to implement it in a programmable logic array (PLA) network arises if the number of variables of the function is greater than the number of inputs of the array. The operation of decomposition requires selection of a solution path leading to an optimal result, i.e., a PLA scheme should be generated which is near minimal with acceptable costs of performing the actual computation process.

Generation of the optimal result requires decomposition of a function allowing its implementation with the smallest number of programmable logic arrays of the required structure.

Some success has been achieved in developing decomposition methods to produce the desired result [1, 2], but the problem cannot be considered solved. The methods which have been developed do not allow analysis of intermediate results during the course of the solution or control of the process. The capabilities of the programmable logic arrays are not fully used, and computations are performed in tabular form.

This article is dedicated to solving the problem of decomposing a logic function with simultaneous computation of a PLA network formula in algebraic form considering the properties of the basic PLA. The function may be assigned by a binary or trinary matrix with one or several inputs.

Suppose the function is given as two sets of values — one M(1) and zero M(0). Given this representation, the following type of decomposition is always possible.

Each set can be arbitrarily divided into a number of subsets:

$$\begin{gathered} M_1(1),\ M_2(1),\ \ldots,\ M_l(1),\ \ldots,\ M_n(1), \\ M_1(0),\ M_2(0),\ \ldots,\ M_l(0),\ \ldots,\ M_n(0). \end{gathered}$$

For any pair, a formula f separating the pair can be found. Then for a subset, for example $M_l(0)$, a set of formulas is found which separates it from all the unit subsets:

$$\begin{gathered} f_l(0)_1,\ f_l(0)_2,\ \ldots,\ f_l(0)_l,\ \ldots,\ f_l(0)_n, \\ \left.\begin{array}{l} M_l(1) \in f_l(0)_i \\ M_l(0) \cap f_l(0)_l = \varnothing \end{array}\right\}. \end{gathered}$$

The logical sum of these formulas separates subsets $M_l(0)$ from set M(1):

$$\left.\begin{array}{l} M(1) \in (f_i(0)_1 \cup f_i(0)_2 \cup \ldots f_i(0)_i \cup \ldots f_i(0)_n \\ M_i(0) \cap (f_i(0)_1 \cup f_i(0)_2 \cup \ldots f_i(0)_i \cup \ldots f_i(0)_n = \varnothing \end{array}\right\}.$$

The intersection of such sums, generated for each subset $M_i(0)$, separates these sets M(0) and M(1)::

$$M(1) \in (f_1(0)_1 \cup f_1(0)_2 \cup \ldots f_1(0)_i \ldots f_1(0)_n)\,(f_2(0)_1 \cup f_2(0)_2 \cup \ldots$$
$$\ldots f_2(0)_i \ldots f_2(0)_n) \ldots (f_n(0)_1 \cup f_n(0)_2 \cup \ldots f_n(0)_n).$$

This formula is implemented by a PLA if it is used as a device for computing the conjunction of the disjunctions (PLA's are generally used to compute the disjunction of conjunctions).

Consequently,

$$\vee (x_1 x_2 \ldots x_i \ldots) = \wedge\, \overline{(\bar{x}_1 \vee \bar{x}_2 \vee \ldots \bar{x}_i \ldots)},$$

where x is $\mathbf{x}$ or $\bar{\mathbf{x}}$.

This type of decomposition and the use of a PLA corresponds to the following computer apparatus[3].

The sets M(1) and M(0) are assigned as matrices, the elements of which $\mathbf{x}$ and $\bar{\mathbf{x}}$ are the values of the variables "1" and "0," respectively.

A matrix is constructed, for example M(0), by inversion of variables in matrix M(0). This matrix covers area M(1) and an area of uncertain values. It has the following properties, which are used to solve the problem at hand.

The matrix $\overline{M(0)}$ is decomposed with respect to the variable x_i:

$$\overline{M(0)} = \left|\begin{array}{c} x_1 x_2 \ldots x_{i-1} \quad x_{i+1} \ldots x_n \\ \cdots\cdots\cdots\cdots \\ x_1 x_2 \ldots x_{i-1}\, \mathbf{x}_i\, x_{i+1} \ldots x_n \\ \cdots\cdots\cdots\cdots \\ x_1 x_2 \ldots x_{i-1}\, \bar{\mathbf{x}}_i\, x_{i+1} \ldots x_n \end{array}\right| = \left|\begin{array}{l} \mathrm{A} \\ \\ \quad \mathbf{x}_i \mathrm{B} \\ \\ \quad \bar{\mathbf{x}}_i\ \mathrm{C} \end{array}\right|, \tag{1}$$

where A, B, C are matrices of lower rank, the corresponding fragments of the initial matrix $\overline{M(0)}$ and generated by removal of element x_i, in particular, A — from the set of rows of $\overline{M(0)}$ not containing element x_i, B — from the set of lines of $\overline{M(0)}$ containing the element $\mathbf{x}_i$, C — from the set of lines of $\overline{M(0)}$ containing element $\bar{\mathbf{x}}_i$.

The matrix $\overline{M(0)}$ is also arbitrarily divided into rows.

Further decomposition of expression (1) with respect to the same variables converts $\overline{M(0)}$ to

$$\overline{M(0)} = \left| \begin{array}{l} x_m \; x_n \; x_p \; \ldots \; D_1 \\ x_m \; x_n \; x_p \; \ldots \quad D_2 \\ \ldots\ldots\ldots\ldots \\ x_m \; x_n \; x_p \; \ldots \qquad D_k \end{array} \right| . \qquad (2)$$

Any element x may be absent. The sign "–," used in trinary matrices, may be omitted, since all variables are represented. Matrix (2) retains all the properties of matrix (1).

Each row fully covers set M(1) and separates from it the corresponding portion of set M(0), while the products of the rows separate M(0) and M(1).

Matrix (2) can always be produced to satisfy the requirements of a PLA: The number of columns is no greater than the number of inputs of a PLA, the number of rows is no greater than the number of terms.

Elements D are similar matrices or one multiple–output matrix in the next network layer.

Subsequent operations with matrices D are simplified, since the matrices separate only the corresponding portions of sets M(1) and M(0).

The form of matrix (2) is changed slightly for a multiple–output function — the number of rows and element D increases, and they are encoded by the output codes of the initial matrix.

Example 1.

$$M(1) = \left| \begin{array}{ccccc} \bar{x}_1 & & \bar{x}_3 & \bar{x}_4 & \bar{x}_5 \\ \bar{x}_1 & \bar{x}_2 & \bar{x}_3 & x_4 & x_5 \\ & & x & \bar{x}_4 & x_5 \\ & & x_3 & x_4 & \bar{x}_5 \\ \bar{x}_1 & x_2 & & \bar{x}_4 & \\ \bar{x}_1 & x_2 & \bar{x}_3 & & \bar{x}_5 \\ x_1 & & x_3 & & x_5 \\ x_1 & & x_3 & x_4 & \\ x_1 & x_2 & x_3 & & \end{array} \right| , \quad \overline{M(0)} = \left| \begin{array}{ccccc} & x_2 & x_3 & x_4 & \bar{x}_5 \\ & x_2 & x_3 & \bar{x}_4 & x_5 \\ & x_2 & \bar{x}_3 & x_4 & x_5 \\ x_1 & & \bar{x}_3 & \bar{x}_4 & \bar{x}_5 \\ x_1 & \bar{x}_2 & & \bar{x}_4 & \bar{x}_5 \\ \bar{x}_1 & x_2 & x_3 & & \\ \bar{x}_1 & & x_3 & x_4 & \\ \bar{x}_1 & & x_3 & & x_5 \end{array} \right| .$$

Matrix M(0) is not presented.

Matrix $\overline{M(0)}$ is decomposed directly with respect to variable x_1:

$$
f \in \overline{M(0)} = \left| \begin{array}{ccc} A & & \\ & x_1 & B \\ & \bar{x}_1 & & C \end{array} \right| ; \qquad A = \left| \begin{array}{cccc} x_2 & x_3 & x_4 & \bar{x}_5 \\ x_2 & x_3 & \bar{x}_4 & x_5 \\ x_2 & \bar{x}_3 & x_4 & x_5 \end{array} \right| ;
$$

$$
B = \left| \begin{array}{cccc} & \bar{x}_3 & \bar{x}_4 & \bar{x}_5 \\ \bar{x}_2 & & \bar{x}_4 & \bar{x}_5 \end{array} \right| ; \qquad C = \left| \begin{array}{cccc} x_2 & x_3 & & \\ & x_3 & x_4 & \\ & x_3 & & x_5 \end{array} \right| .
$$

A similar result is obtained by decomposition with respect to variables x_4 and x_5. Furthermore, matrix $\overline{M(0)}$ is decomposed into three matrices with respect to the rows after removal of the unnecessary letters.

Removing unnecessary letters from matrix C, we obtain $C = \mathbf{x}_3$, and consequently,

$$
f = \left| \begin{array}{ccc} A & & \\ & x_1 B & \\ & \bar{x}_1 & x_3 \end{array} \right| .
$$

Thus, implementation of this function requires not four, but only three PLA's (4, 1, 3).

Example 2. A multiple-output function assigned by a trinary matrix which requires PLA's (6, 5, 20):

$$
M_0 = \left| \begin{array}{ccc|ccc|cc|cc|ccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 & x_{10} & y_1 & y_2 & y_3 \\ 0 & - & 0 & - & - & 0 & 0 & - & - & 0 & 1 & 0 & 0 \\ 0 & - & 0 & - & 0 & 0 & - & - & - & 0 & 1 & 1 & 0 \\ - & 1 & 0 & - & 0 & 0 & - & - & - & 0 & 1 & 0 & 1 \\ - & 0 & 0 & 0 & - & 0 & - & 0 & 0 & - & 1 & 0 & 1 \\ 1 & - & 0 & 0 & - & 0 & - & 0 & 0 & - & 0 & 0 & 1 \\ - & 0 & - & 0 & - & - & - & - & 0 & 0 & 0 & 1 & 1 \\ 1 & - & - & 0 & - & - & - & - & 0 & 0 & 0 & 1 & 1 \end{array} \right| \begin{array}{c} \\ (1) \\ (2) \\ (3) \\ (4) \\ (5) \\ (6) \\ (7) \end{array}
$$

Matrix $\overline{M(0)}$ can be written combined for all three outputs, its expansion with respect to two variables into sequences $\mathbf{x}_7$-$\mathbf{x}_5$-$\mathbf{x}_4$, $\mathbf{x}_9$, leading to the solution

$$
M_1 = \left| \begin{array}{ccccc|cc} x_7 & B_7 & & & & y_2 & y_3 \\ & B_7 & x_5 & & & & y_3 \\ & & x_5 & B_5 & & y_2 & \\ & & & & A_5 & y_2 & \end{array} \right| ,
$$

$$
M_2 = \left| \begin{array}{cccc|cc} x_4 & x_9 & B_4 & & A_5 & \\ x_4 & x_9 & & B_9 & & y_1 \end{array} \right| ,
$$

$$
M_3 = \left| \begin{array}{cccccc|cccc} x_1 & & x_3 & x_6 & & x_{10} & B_7 & & & \\ & \bar{x}_2 & x_3 & x_6 & & x_{10} & & B_5 & & \\ & x_2 & x_3 & x_6 & x_8 & & & & B_4 & \\ \bar{x}_1 & & x_3 & x_6 & x_8 & & & & B_4 & B_9 \\ & x_2 & & & & x_{10} & & & & B_9 \\ \bar{x}_1 & & & & & x_{10} & & & & B_9 \end{array} \right| .
$$

Thus, the assigned function is implemented by three rather than four PLA's. Two of them (M_1 and M_2) are not fully utilized as to inputs, all three are incompletely utilized as to outputs.

Obviously, $\overline{M}$ can be simplified if we used the one value rows rather than the zero value rows to output y_2. Then, the third and fourth rows of matrix M_0 are not included in matrix $\overline{M}$.

Conclusions. A method is suggested for defining a PLA network by algebraic transformations of the initial function, fixed in tabular form, to a network formula. The problem is solved of implementing the network in PLA's, with smaller number of inputs than the number of variables in the function. The method is applicable for both binary and trinary one- and multiple-output functions. The method allows analysis and adjustment of the solution during the process of computation of the network formula. This allows expedient search assuring the most complete utilization of the base PLA's as to inputs, terms and outputs.

The result is a reduction in the number of base PLA's and simplification of the solution process.

Where the functions are assigned by a trinary matrix, the need to orthogonalize the rows is eliminated.

1. Suprun, V. P. "Method of Implementing Boolean Functions Fixed In Interval Form In PLA's." Avtomatika i vychisl tekhnika, 1982, No. 5, pp. 30-36.

2. Dyachenko, Yu. G. "Implementing Sequential Automata In a Network of PLA's of Limited Size." USiM, 1985, No. 6, pp. 69-73.

3. Lazarev, V. I. "Matrix Method of Converting Binary Logic Function." Avtomatizatsiya protsessov v selkhozmashinostroyenii (Automation of Processes In Agricultural Machine Building). Rostov n/D, Institute of Agricultural Machine Building, 1977, pp. 45-52.

UDC 681.3.06

Organization of Parallel Computation and Development of Program Packages Based on Parallel Acting System[1]

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian (manuscript received 12 Jun 87) No 3, May–Jun 1988, pp. 24–29

[Article by L. A. Bulysheva, V. M. Fomin, R. T. Chernysheva and N. N. Yanenko, deceased]

[Text] **Introduction.** The modeling of complex physical processes and the solution of many great national economic problems are related to further development of computer technology and system programming. The solution of these problems will be greatly facilitated by the creation of multiprocessor computer systems. The speed of computation on such systems can be increased approximately in proportion to the number of processors in the system, if the algorithm used allows calculations to be performed simultaneously by all processors.

As computer systems are brought into practical application, problems of organizing parallel computation become more pressing, since parallel computation is one means of accelerating the solution of problems[1-3]. The first works on parallel computing appeared in the early 1960's and largely defined computing practice of the day. By the mid 1960's a theory of parallel programming had been developed, using the results of scheduling theory, graph theory, general algebra, mathematical logic and other sciences[4-11]. In the 1970's, a number of multiprocessor computers with parallel computation were created, including the STARAN, ILLIAC-IV, B6700, Star-100, Burroughs, Cray-1, Elbrus-1, Rere, M-10, PS-2000 and others[12].

There is an extensive library of works on the theory of parallel programming. Due to the limited space available to us, we have not set ourselves the task of reviewing the contemporary state of affairs in this area. Our purpose is rather to study the problems of organizing parallel computation based on engineering and physical application program packages. In our view, these problems are important because the development of parallel algorithms was begun comparatively recently, and the available fund of programs is based on sequential organization of computation. It is no less important that available engineering and physical application program packages were created using functional modules based on sequential algorithms.

Major trends in the theory of parallel programming. We can briefly distinguish the following problems related to the theory of parallel programming:

— The general or formal theory of parallel computation[13-16];

— parallel programming methods and languages[17-32];

[1]This article is the last work written by a well-known mathematician, academician, USSR Academy of Sciences, Nikolay Nikolayevich Yanenko (1925–1983). At the request of his coauthors, the editors have studied the article and concluded that although the review does not present new ideas, its basic ideas concerning the organization of application software packages and parallel computations are still important.

Stylistic corrections were made to the article as it was prepared for printing.

— operational problems of parallel programming[33-42];

— creation of automatic parallel generators allowing conversion of sequential programs to parallel programs[43-59].

We note that the organization of parallel computation, i.e., the paralleling problem, is complex[8]. In general, by paralleling we mean decomposition of a program (task) so that individual fragments can be executed on different processors, such that parallel execution yields the same results as would be generated by sequential execution[7, 8, 53]. Complete solution of the paralleling problem is possible only in a few areas — the creation of highly productive computers (multiprocessor computer systems), development of new methods for solving large problems, paralleling of existing algorithms[2, 56, 57], creation of effective parallel algorithms and programs[15], plus also continued development of system programming[7, 13] (theory of structural parallel programming, programming languages, implementation of automatic paralleling[49, 50], parallel programming systems[43]), and finally, the creation of application program packages for parallel processors.

The generation theory of parallel computation is directed toward the investigation of the properties of parallel computation, construction of formal models of parallel programs and systems, allowing investigation of parallelism, asynchronism, equivalence and other properties of parallel computations.

Methods of parallel programming are used to create practical parallel programming languages, usually considered the final versions of parallel languages oriented toward various classes of systems and problems, or as modifications of existing languages with the addition of facilities for assigning parallelism. Thus, many works have suggested expansion of languages such as ALGOL-60 and FORTRAN by the addition of facilities for explicit assignment of parallelism, i.e., operators such as fork, join, parallel for, etc. Facilities for separation of parallel branches are provided in some algorithmic languages (*PL/1,* ALGOL-68), but are included only as necessary (for more detail, cf.[7, 13, 33, 41, 42, 53, 55]).

The problem of paralleling (i.e., the creation of automatic parallel program generators, the development of algorithms for conversion of sequential programs to parallel programs) is one of the most important problems in parallel programming. This is because there are many sequential algorithms and programs which should not be abandoned.

Most interesting for the authors are problems of paralleling programs and operational problems of parallel programming. The operational problems we allude to include determination of certain numerical characteristics of parallel computation, for example the expected time of execution of problems and the number of processors required. Operational problems of parallel programming thus include the solution of problems of composition of a schedule for execution of programs or program modules on multiprocessor computers.

As a rule, the models of programs used to investigate operational problems are graphs. The points of the graphs are assigned weights which correspond to the times of execution of individual modules, programs or operators. Network methods are used to determine the minimum execution time of a program (computational chain), the time when execution can start for each program (operator) — the earliest moment and the latest moment of execution, as well as the number of processors necessary to execute a parallel program in a certain fixed time, i.e., the minimum number of processors required is determined. If the execution of a sequential computational chain is a control graph with informational communications between the points of the graph, while a parallel program is represented as an information graph limited by a set of control connections[59], the task of paralleling is reduced to removing the controlling limitations from the information graph[7, 13].

The architect of multiprocessor systems largely determines both the programming style and the selection of numerical methods. After an acceptable solution method is found, the problem arises of its implementation in some practical programming language, i.e., the problem of developing linguistic facilities for writing the parallel algorithms. The possibility of organizing parallel information processing appeared with the development of computers which can use several devices to execute a single task.

The following levels of parallelism are distinguished:

— Microparalleling (parallelism within programs at the level of expressions);

— miniparalleling (at the level of operators);

— macroparalleling (parallelism among programs at the level of tasks).

By analogy, three levels of paralleling of data are used, although 1-to-1 correspondence between program and data parallelism levels has not been achieved:

— Microparalleling (at the level of elementary data);

— miniparalleling (at the level of arrays);

— macroparalleling (at the level of data structures).

Paralleling of sequential programs can be performed by two means: Dynamically — during the course of computations, and statistically — during translation.

With dynamic paralleling, analysis of parallelism and determination of independent branches are performed during computation — this is paralleling at the operating system level. With statistical paralleling, a source program is translated to a program in a parallel programming language before it is executed — this is paralleling at the translator level. It requires special system paralleling facilities or may be performed in the process of translation from source languages.

The following programming automation facilities are presently available for the creation of parallel programs[41]:

— A multilevel structural parallel programming automation system, oriented toward the creation of parallel system programs, called multiprocessing[14, 15];

— languages for writing parallel algorithms, either expansions of existing languages (ALGOL, FORTRAN) or new parallel programming languages designed for the creation of interactions among parallel program branches[17, 21, 22, 32];

— debugging facilities allowing modeling in a single processor of programs created for implementation on several processors;

— specialized algorithm parallelers, allowing paralleling of acyclic algorithms and construction of parallel programs with predetermined numbers of branches[50];

— methods of paralleling programs with loops, without loops[18, 19], and linear program structures[28, 29, 33, 42];

— splitting and clearing methods[59] and many others[2, 60].

All of this indicates the importance of paralleling and the need to evaluate the effectiveness of solution of problems on multiprocessor computers. Let us discuss this question briefly.

Effectiveness of parallel computation. The effectiveness of solving problems in parallel mode has been repeatedly evaluated both here and abroad. The effectiveness of solving problems in parallel mode is closely related to the general problem of defining computer effectiveness.

The statement of the problem might be as follows. An algorithm for a task is subdivided into several branches which run simultaneously on different processors. The purpose of the computation is to reduce the time required to perform an individual task on the system. With limited computer resources, construction of parallel computation algorithms with high effectiveness is a complex task.

The comparison and evaluation of algorithms utilizes[53] the acceleration factor K_y and the effectiveness factor $K_э$:

$$K_y = T_1 / T_n,$$

where T_1 is the time required to perform a task on a single-processor computer; T_n is the time required to perform the same task on an n-processor computer;

$$K_э = K_y / n,$$

where n is the number of processors.

It has also been suggested that the effectiveness of executing a program be evaluated based on the processor loading factor[54]:

$$P = \Sigma t / n \cdot t,$$

where P is the probability that at a certain moment in time an arbitrarily selected processor will not be idle (will be executing a program instruction); Σt is the total time of operation of the processors during execution of a program; T is the total time required to execute the program; n is the number of processors.

A method was suggested in[36] for evaluating the effectiveness of solution of a problem in parallel form. To do this, the cost of performing a certain task in parallel mode was compared with its performance in ordinary mode. Analytic expressions were derived for the cost of performing operations in parallel mode and the effect achieved by paralleling was determined directly in the process of operations using data generated by the system after completion of the task. The conclusions generated confirm the practice of paralleling computations. Thus, most authors believe that paralleling is suitable for tasks requiring large quantities of random-access memory with little or moderate exchange of data with peripheral devices; for average tasks the expediency of paralleling is determined by the number of exchanges between parallel program branches; small tasks should not be paralleled regardless of the number of exchanges of data with peripheral devices.

Equations for evaluating the effectiveness of paralleling were presented in[38].

We know that the main task of mathematical technology is optimization of a computer chain[1]. This task is particularly important for parallel computations.

All of the problems we have listed have been discussed at length in the literature, except for problems related to engineering and physical program packages for multiprocessor computers.

Some problems of application software package development. Application software packages are widely used for planning and in scientific research[61]. Among the various types of packages, great attention has been given to the development of engineering-physical packages[40, 62, 65] designed to automate numerical simulation of complex physical processes and planning of engineering-technical structures. The basic principles of organizing such packages and their software configuration have been developed, with two basic configurations: System and functional[40, 62]. The structural unit (functional module) of an engineering-physical package is the application program, implementing a certain physical process in numerical form[40, 63].

An example of construction of an engineering-physical package for computation of "flow around a body" based on the ISTOK system and available application programs as functional modules is presented in[66]. The system configuration used is a modified version of ISTOK-1, invariant to the class of problem solved. Therefore, ISTOK allows finished software products to be included in the functional implementation by their formal representation[41, 65, 66].

The experience of using batch technology to solve problems related to flow around a body has shown that such problems can be successfully solved using batch technology, using application programs as functional modules. Effective performance of calculations requires organization of parallel computation, which results from the large volume of main memory occupied by each module and the duration of the computations, i.e., the need to reduce the time of computation required to construct the geometry of the body. The results of calculations are presented in[66].

The computing process in the package can be represented in a formalized manner by stating that it is constructed on the basis of automatic construction of computational chains such as

$$M_i \Rightarrow M_1 \Rightarrow M_2 \Rightarrow \ldots \Rightarrow M_m;\ i=1, \ldots, m,$$

where M_i is the functional configuration modulus; i is the modulus number of the system control program[62-65]. Looping computation is also possible as follows:

$$\begin{tikzcd}
{} \arrow[r, Rightarrow] & M_1 \arrow[r, Rightarrow] & M_2 \arrow[r, Rightarrow] \arrow[l, bend left] & \cdots \arrow[r, Rightarrow] & M_i \arrow[r, Rightarrow] \arrow[loop above] & \cdots \arrow[r, Rightarrow] & M_m \arrow[r, Rightarrow] & {} \arrow[llllll, bend left]
\end{tikzcd}$$

A module in the engineering-physical package can be formally represented as

$$M_i = M_i(F_i,\ A_i,\ R_i,\ Q_i),$$

where M_i is the name of module i; F_i is the functional purpose of the module; A_i and R_i are sets of arguments and results, respectively; Q_i is the set of variable quantities, acting simultaneously as arguments and results of the modulus, where

$$A_i=\{a_{ijk}\};\ R_i=\{r_{ijk}\};\ Q_i=\{q_{ijk}\}$$

or

$$A_i=\{a_{i11}, a_{i22}, \ldots, a_{ipn}\},\quad R_i=\{r_{i11}, r_{i22}, \ldots, r_{ipn}\},$$
$$Q_i=\{q_{i11}, q_{i22}, \ldots, q_{ipn}\},$$

where i is the set of modules in the engineering-physical package, $i=1, \ldots, m$; j is the set of parameters of module i, $j=1, \ldots, p$; k is the set of values of parameter p of module i, $k=1, \ldots, n$.

The modules of the engineering-physical package are interrelated through their data (information) and in control, so that the results of previous modules serve as arguments for subsequent modules. The parameters of the functional engineering-physical package modules are such that

$$\bigcup_{p=i+1}^{m}\left(A_i \bigcap_{i<p} R_p\right)=\varnothing,\quad i=1, \ldots, m-1.$$

Consequently, sequential computation chains cannot be effectively implemented in multiprocessor computers unless special facilities for their paralleling are developed. The architecture of engineering-physical packages developed for single-processor computers does not decrease the time of modeling of objects in multiprocessor computer systems.

Architecture of engineering-physical packages based on parallel action systems. There are several possible means for organizing paralleling of computations in application program packages:

— Organize parallel computations of several versions of the same problem (module);

— organize paralleling, optimizing the computational chain consisting of modules designed for solution of a certain physical problem.

There are two levels of paralleling in engineering-physical packages:

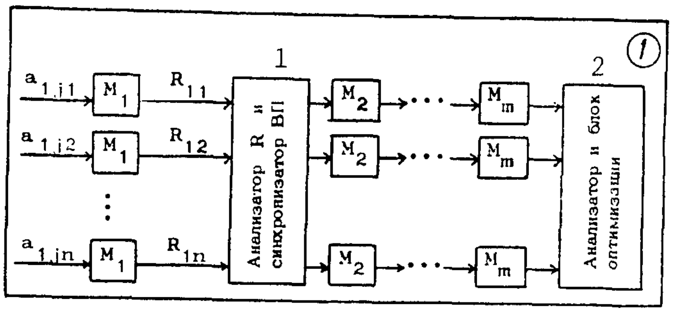
— Paralleling with respect to parameters of a single module of an optimal computational chain, which is used primarily for iterational processes;

— paralleling by optimizing a computational chain: Conversion from a sequential to a parallel chain.

It is thought that paralleling is suitable for tasks occupying all of computer memory or requiring minimization of computing time.

A principle for paralleling of engineering-physical package computations has been suggested and the basic changes to be introduced to the system configuration outlined. The basic purpose of the engineering-physical package is numerical modeling of an object under various fixed conditions[61], for example when argument $a_{ij} \in A_i$ changes continually through a finite interval $[a_{ij}^{(1)}, a_{ij}^{(2)}]$ or takes on discrete values from a list: $a_{ij}=\{a_{ij1}, a_{ij2}, \ldots, a_{ijn}\}$.

As we can see, the very nature of numerical modeling allows organization of parallel computations. Let us assume that the number of processors n, $n \leqslant L$ (L is the number of versions). Computations are organized as follows: First computations by module M_1 are organized n times on different processors with argument $a_{ij}=\{a_{ij1}, \ldots, a_{ijn}\}$, respectively, and n sets of results $R_1=\{R_{11}, R_{12}, \ldots, R_{1n}\}$ are produced. An intermodular interface is then organized between modules M_1 and M_2; data are automatically transmitted with the corresponding identification among the processors and module M_2 is called. The parameters required by the planner are stored in a data base. The same communications are provided between modules M_2 and M_3, etc. until all links of the chain are covered (Figure 1).



Key: 1, R — Analyzer and computing process synchronizer; 2, analyzer and optimizer.

We should note that this organization of computations requires, after one module has executed n times, analysis of the results obtained (their comparison, combined processing, selection, etc.). Whereas when a module runs but once this stage is merely desirable, for parallel computations it is absolutely obligatory due to the expansion in the volume of data. Let us assume that after some analysis, n computational results were found to be unacceptable; the problem then arises of synchronizing the computing process (module M_2 runs n_1 times, module M_1 runs $(n-n_1)$ times or module M_1 runs n times, module M_2 then runs $(n+n_1)$ times, etc.). Synchronization is also possible when the numerical algorithm implemented in a module depends on a_{ij}, for example in iterational algorithms.

Therefore, certain changes must be made to the architecture of an engineering-physical package for organization of parallel computations:

1) Functions of the basic system components of the package must be expanded: Introduce addition operators for data base access, include in the input language of the package operators for assignment of parallel computations, increase the load on the package monitor, introduce a new component — the paralleling unit, organizing parallel computation by optimizing computational chains or interactions among machines;

2) a description of a module must be developed — a certificate, including the information required to organize parallel computations;

3) create a package supervisor, responsible for synchronizing computations.

The "flow around a body" package[66] can serve as an example of an application software package for which the principle of parallel computations outlined above can be applied.

The library of functional modules of this package consists of:

— Module M_1 for computation of supersonic flow around three-dimensional bodies with blunted head portion by an ideal gas stream;

— module M_2 for computation of three-dimensional supersonic flow around a conical nose portion by an ideal gas stream;

— module M_3 for computation of flow around bodies by the Marsh method.

The mathematical statement of the problem and numerical algorithms implementing these modules are studied in[67].

Let us note the peculiarities of the modules required to organize computations. The application software package of[66] was created to determine the aerodynamic coefficients and moments for a body of arbitrary shape over a broad range of Mack numbers (**M**) and angles of attack (α). The machine time required to calculate the aerodynamic parameters is quite considerable.

These computations can easily be implemented in a computer system using the plan suggested above, generating results in a much shorter time. Actually, suppose

$$M_1 = M_1(A_{11}, A_{12}, A_{13}, R_{11}, R_{12}, R_{13}, A_{14}),$$

where A_{11} is the Mack number of the incident stream; A_{12} is the angle of attack; A_{13} are the physical characteristics of the medium; A_{14} are the geometric parameters of the blunt bodies; $R_{11}=\{\rho, \mathbf{u}, T\}$ are the three-dimensional solution arrays; ρ is the density of the medium; $\mathbf{u}$ is the velocity vector; T is the temperature of the medium; R_{12} are the integral parameters of the medium flowing around the body (including aerodynamic coefficients and moments); R_{13} is the array of coordinates of the difference grid (spherical system of coordinates).

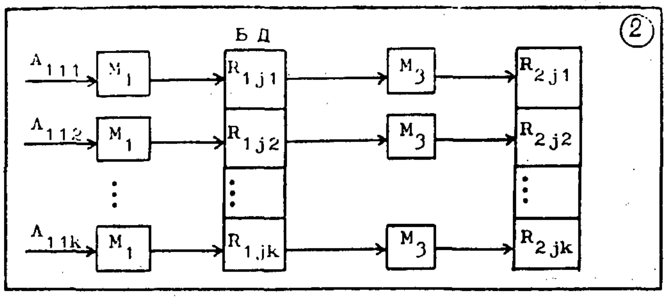
Module M_3 is

$$M_3 = M_3(A_{31}, A_{32}, A_{33}, A_{34}, A_{35}, R_{31}, R_{32}, R_{33}),$$

where $A_{31}=A_{11}$, $A_{32}=A_{12}$; $A_{32}=A_{13}$; A_{34} represents the geometric parameters of the body; A_{35} represents the two-dimensional arrays of the solution for a fixed cross-section x, generated either by module M_1 or module M_2; R_{31} are two-dimensional arrays of solutions for cross section $x=x^{n+1}=x^n+(\Delta x)^n$; R_{32} are the integral characteristics of the medium flowing around the body where $x=x^{n+1}$; R_{33} is the array of two-dimensional difference grid coordinates.

The scheme for parallel computations for this application package is shown in Figure 2.

As we can see from this Figure, solution of the problem of determining the aerodynamic parameters of a body as a function of **M** number is performed as follows: First module M_1 runs n times (equal to the number of processors), producing various output data required for the operation of module M_3, then parallel organization of the third module is organized, for which



the system configuration organizes the intermodular interface, indicating the names of the parameters.

Thus, when a multiprocessor computer system is used, the time required to solve the problem of determining $C_x = C_x(\alpha, M)$ can be reduced in proportion to the number of processors.

Conclusions. Problems of organizing parallel computations are of great practical significance. An extensive fund of sequential programs and algorithms has been accumulated, which must be used effectively on multiprocessor computers. To do this, new methods for paralleling programs, algorithms and operators are now being developed, plus automatic parallelers which can analyze programs and introduce parallelism. New parallel programming languages are being developed and previously created languages expanded by inclusion of parallel computing facilities. Of greatest interest are problems of paralleling application software packages. It is their development which brings up the entire complex range of problems related to paralleling computations in parallel programming.

We should also note that we have not discussed in this work problems relating to the development of parallel algorithms, since these problems have been given great attention in the literature, for example in works on analysis of algorithms[28, 29, 60]. Our desire here was to emphasize that the construction of software configurations and the technology of programming using computer systems are no less complex and cumbersome tasks than the creation of parallel algorithms.

REFERENCES

1. Yanenko, N. N. "Problems of Modular Analysis and Parallel Computation In Mathematical Physics Tasks." Parallelnoye programmirovaniye i vysokoproizvoditelnye sistemy (Parallel Programming and Highly Productive Systems). Novosibirsk: Computer Center, Siberian Division, USSR Academy of Sciences, 1980, Part 1, pp. 135–144.

2. "Organization of Parallel Computation and 'Paralleling' of Computation." N. N. Yanenko, A. N. Konovalov, A. M. Bugrov and G. V. Shustov. Chislen metody mekhaniki sploshnoy sredy. 1978, 9, No. 7, pp. 139–146.

3. Korolev, L. N. Struktury EVM i ikh matematicheskoye obespecheniye (Computer Structures and Their Software). Moscow, Nauka Press, 1978, 351 pp.

4. Glushkov, V. M., Tseytlin, G. Ye., Yushchenko, Ye. L. Algebra. Yazyki. Programmirovaniye (Algebra. Languages. Programs). Kiev, Nauk. dumka Press, 1978, 320 pp.

5. Dal, U., Deykstra, E., Khoor, K. Strukturnoye programmirovaniye (Structured Programming). Moscow, Mir Press, 1975, 248 pp.

6. Yershov, A. P. Vvedeniye v teoreticheskoye programmirovaniye (Introduction to Theoretical Programming). Moscow, Nauka Press, 1977, 288 pp.

7. Kotov, V. Ye. "Theory of Parallel Programming. Applied Aspects." Kibernetika, 1974, No. 1, pp. 1-16; No. 2, pp. 1-18.

8. Kotov, V. Ye. Rasparallelivaniye programm i algoritmov i arkhitektura vysokoproizvoditelnykh EVM (Paralleling of Programs and Algorithms and Architecture of Highly Productive Computers). Moscow, 1981, 54 pp. (Preprint/National Institute of Scientific and Technical Information; No. 5).

9. Kotov, V. Ye. Vvedeniye v teoriyu skhem programm (Introduction to Program System Theory). Novosibirsk, Nauka Press, 1978, 257 pp.

10. Zhirov, V. F. Matemeticheskoye obespecheniye i proektirovaniye struktur EVM (Software and Planning of Computer Structures). Moscow, Nauka Press, 1979, 159 pp.

11. Conway, R. V., Maxwell, V. L., Miller, L. V. Teoriya raspisaniy (Scheduling Theory). Moscow, Mir Press, 1975, 360 pp.

12. Golovkin, B. A. Parallelnye vychislitelnye sistemy (Parallel Computer Systems). Moscow, Nauka Press, 1980, 519 pp.

13. Narinyani, A. S. "Theory of Parallel Programming. Formal Models." Kibernetika, 1974, No. 3, pp. 1-15, No. 5, pp. 1-14.

14. Glushkov, V. M., Velbitskiy, I. V. "Programming Technology and Problems of Its Automation." USiM, 1976, No. 6, pp. 75-93.

15. Glushkov, V. M., Tseytlin, G. Ye., Yushchenko, Ye. L. Problemy analiza i sinteza strukturirovannykh parallelnykh programm (Problems of Analysis and Synthesis of Structured Parallel Programs). Kibernetika, 1981, No. 3, pp. 1-17.

16. Niriyev, R. M. "Necessary and Sufficient Conditions for Significant Paralleling of Programs In Loops." Tekhn. kibernetika, 1976, No. 2, pp. 105-111.

17. Burgin, M. S., Burgina, Ye. S. "Subdivision In Languages and Parallel Computations." Programmirovaniye, 1982, No. 3, pp. 10-21.

18. Valkovskiy, V. A. "Parallel Execution of Loops. The Method of Parallelepipeds." Kibernetika, 1982, No. 2, pp. 51-62.

19. Doroshenko, A. Ye. "Conversion of Cyclical Operators To Parallel Form." ibid, No. 4, pp. 22-28.

20. Valkovskiy, V. A. "Method of Loading Several Processors With a Single Task." Sistemnoye i teoreticheskoye programmirovaniye (System and Theoretical Programming).

Novosibirsk: Computer Center, Siberian Division, USSR Academy of Sciences, 1974, pp. 116–129.

21. Kleshchev, A. S. "Relational Programming Language and Principles of Its Implementation In Sequential Computers." Programmirovaniye, 1981, No. 6, pp. 45–53.

22. Konstantinov, V. I., Sviridenko, D. I. "Mathematical Semantics of a Parallel Programming Minilanguage." Probl. programmirovaniya, [Problems of Programming] Novosibirsk: Computer Center, Siberian Division, USSR Academy of Sciences, 1976, pp. 81–93.

23. Kotov, V. Ye. O parallelnykh yazykakh (On Parallel Languages). Novosibirsk, 1979, 49 pp. (Preprint, Siberian Division, USSR Academy of Sciences, Computer Center, No. 145).

24. Mirenkov, N. N., Simonov, S. A. "Determining Parallelism In Loops By Simulating Their Execution." Kibernetika, 1981, No. 3, pp. 28–33.

25. Kartsev, M. A. "Principles of Organizing Parallel Computations, Computing System Structures and Their Implementation." ibid, No. 2, pp. 68–74.

26. Parallelnoye programmirovaniye i vysokoproizvoditelnye sistemy (Parallel Programming and Highly Productive Systems). Edited by G. I. Marchuk and V. Ye. Kotov. Novosibirsk: Computer Center, Siberian Division, USSR Academy of Sciences, 1980, 290 pp.

27. Mirenkov, N. N. "Structural Parallel Programming." Programmirovaniye, 1975, No. 3, pp. 3–14.

28. Faddeyev, D. K., Faddeyeva, V. N. "Parallel Computing In Linear Algebra." Kibernetika, 1977, No. 6, pp. 28–40.

29. Faddeyeva, V. N., Faddeyev, D. K. "Parallel Computer In Linear Algebra," ibid, 1982, No. 3, pp. 18–31.

30. Todorov, G. Al. "Analysis of Parallelism of Computing Processes In Looping Program Structures." Programmirovaniye, 1980, No. 1, pp. 11–23.

31. Korablin, Yu. P. "Problem of Correctness of Graph Diagrams of Parallel Algorithms." ibid, 1978, No. 5, pp. 45–53.

32. Kutepov, V. P., Korablin, Yu. P. "A Graph-Plan Parallel Algorithm Language." ibid, No. 1, pp. 3–11.

33. Khalilov, A. I. "Some Problems of Multiprocessing." Kibernetika, 1973, No. 1, pp. 103–114.

34. Velbitskiy, I. V. "Programming Technology — Some Results and Prospects for Development." Novosibirsk: ITPM, Siberian Division, USSR Academy of Sciences, 1981, 49 pp.
35. Kulbak, L. I., Karaban, D. I., Prokhorenko, S. S. "Characteristics for Estimating Reliability of Multiprocessor Computer Systems." Avtomatika i vychisl. tekhnika, 1982, No. 1, pp. 67–71.

36. Konoshenko, M. P. "Estimating Effectiveness of Solution of a Problem In Parallel Mode." Programmirovaniye, 1982, No. 5, pp. 80-87.

37. Kovalenko, N. S. "One Model of a Parallel Computing System." Parallelnoye programmirovaniye i vysokoproizvoditelnye vychislitelnye sistemy (Parallel Programming and Highly Productive Computer Systems). Novosibirsk, Computer Center, Siberian Division, USSR Academy of Sciences, 1980, Part 2, pp. 145-151.

38. Krinitskiy, N. A., Chernova, T. F. "Simulation of Operating Systems." Programmirovaniye, 1981, No. 3, pp. 77-85.

39. Petrov, P. A., Nikolayev, A. V. "Analytic Model for Estimating Throughput of a Multiprocessor Computer System." Avtomatika i vychisl. tekhnika, 1982, No. 4, pp. 63-65.

40. "The Current Status, Means for Further Development of Packages and Architecture In the ISTOK System for the Organization and Utilization of Application Program Packages." N. N. Yanenko, V. M. Fomin, R. T. Chernysheva, et. al. Printsipy organizatsii paketov prikladnykh programm: Tez. dokl. (Principles of Organizing Application Program Packages: Abstracts of Reports). Novosibirsk: ITPM, Siberian Division, USSR Academy of Sciences, 1981, pp. 15-17.

41. Valkovskiy, V. A. Lektsii po parallelnomu programmirovaniyu (Parallel Programming Lectures). Novosibirsk: Computer Center, Siberian Division, USSR Academy of Sciences, 1982, 82 pp.

42. Pentkovskiy, V. M. Avtokod Elbrus. Printsipy postroyeniya yazyka i rukovodstvo k polzovaniyu (Elbrus Autocoder. Language Design Principles and Users Guide). Moscow, Nauka Press, 1982, 351 pp.

43. Gritsay, V. P., Tseytlin, G. Ye. "Some Problems of Automating Structured Parallel Programming." Kibernetika, 1979, No. 1, pp. 106-111.

44. Stolyarov, L. N. "Structural Analysis of Algorithms for Effective Organization of Computing." Vychislitelnye sistemy i avtomatizatsiya nauchnykh issledovaniy (Computer Systems and Automation of Scientific Research). Moscow, Nauka Press, 1980, pp. 23-45.

45. Valkovskiy, V. A. "Automation of Parallel Programming." Parallelnoye programmirovaniye i vysokoproizvoditelnye sistemy (Parallel Programming and Highly Productive Systems). Novosibirsk: Computer Center, Siberian Division, USSR Academy of Sciences, 1980, pp. 15-19.

46. Vysokoproizvoditelnye vychislitelnye sistemy: Tez. dokl. Vsesoyuz soveshchaniya (Highly Productive Computer Systems: Abstracts of Reports of National Conference) (Tbilisi, September 1981). Moscow, 1981, 218 pp.

47. "Plan for the 'Multiprotsessist' Automated Structured Parallel Programming System." V. P. Gritsay, G. T. Konovalov, L. M. Melen, et. al. Parallelnoye programmirovaniye i vysokoproizvoditelnye sistemy (Parallel Programming and Highly Productive Systems). Novosibirsk: Computer Center, Siberian Division, USSR Academy of Sciences, 1980, pp. 104-106.

48. Kazmin, A. I., Menn, A. A., Nepeyvoda, N. N. "Tabular Approach to Automatic Program Synthesis." Programmirovaniye, 1982, No. 2, pp. 24-34.

49. "Automation of Seminatural Modeling on Multi-Machine Systems (Software Aspects)" A. I. Kazmin, A. A. Menn, N. V. Ponomarev, V. N. Popolitov. Avtomatika i telemekhanika, 1982, No. 7, pp. 157-165.

50. Kerbel, V. G. "Implementation of Experimental Algorithm Paralleler." Vychisl. sistemy, 1979, No. 78, pp. 54-69.

51. Levin, V. I. "Logical Method for Optimizing Sequence of Solution of Problems In Computer Systems." Avtomatika i vychisl. tekhnika, 1982, No. 3, pp. 55-61.

52. Lavrov, S. S., Zalogova, L. A., Petrushina, T. I. "Principles of Planning the Solution of Problems In an Automatic Program Synthesis System." Programmirovaniye, 1982, No. 3, pp. 35-44.

53. Molchanov, I. N. O nekotorykh problemakh rasparallelivaniya pri reshenii nauchno-tekhicheskikh zadach (Some Problems of Paralleling In the Solution of Scientific-Technical Problems). Novosibirsk: ITPM, Siberian Division, USSR Academy of Sciences, 1981, 54 pp.

54. Stepanov, N. V., Chumakov, M. V. "Effectiveness of Program Parallelism Utilization In Computer Systems." Mnogoprotsessornye vychisl. struktury (Multiprocessor Computing Structures). 1980, No. 2, pp. 68-70.

55. Khalilov, A. I. "The Problem of Paralleling Programs." Probl. kibernetiki, 1974, No. 28, pp. 157-178.

56. Abramov, V. M. "Analysis of Algorithms To Parallel Computations." Vychislitelnye sistemy i avtomatizatsiya nauchnykh issledovaniy (Computing Systems and Automation of Scientific Research). Moscow, Institute of Physics and Technology, 1980, pp. 53-63.

57. Belousov, A. I. "The Problem of Paralleling Algorithms." Programmirovaniye, 1978, No. 5, pp. 53-62.

58. Mityayeva, S. A. "The Problem of Paralleling Programs For Multiprocessor Computing Systems." Kibernetika, 1980, No. 2, pp. 47-50.

59. Itkin, V. E. "Two Methods of Paralleling Programs." Teoriya i praktika sistemnogo programmirovaniya (Theory and Practice of System Programming). Novosibirsk Computer Center, Siberian Division, USSR Academy of Sciences, 1977, pp. 95-109.

60. Konovalov, A. N., Yanenko, N. N. "Some Problems of the Theory of Modular Analysis and Parallel Programming For Problems of Mathematical Physics and the Mechanics of Continuous Media." Sovremennye problemy matematicheskoy fiziki i vychislitelnoy matematiki (Contemporary Problems of Mathematical Physics and Computational Mathematics). Moscow, Nauka Press, 1982, pp. 135-141.

61. Printsipy razrabotki paketa prikladnykh programm na baze nestandartizovannogo funktsionalnogo napolneniya (Principles of Development of Application Program Packages Based on Nonstandardized Functional Configurations). L. P. Bass, T. A. Germogenova, V. I. Zhuravlev, et. al. Moscow: Institute of Applied Mathematics, USSR Academy of Sciences, 1979, 29 pp.

62. Karpov, V. Ya., Koryagin, D. A., Samarskiy, A. A. "Principles of Development of Packages for Mathematical Physics Tasks." Zhurn. vychisl. matematiki i mat. fiziki, 1978, 18, No. 2, pp. 458–467.

63. "Modular Analysis of a Program System." N. N. Yanenko, Ya. F. Savchenko, T. A. Bondar, et. al. Chislen. metody mekhaniki sploshnoy sredy (Numerical Methods of Mechanics of Continuous Media). 1975, 6, No. 4, pp. 128–139.

64. Sistema obespecheniya kompleksov programm matematicheskoy fiziki "System to Support Mathematical Physics Program Systems." A. V. Voronkov, V. I. Zimenkov, V. I. Legonkov, et. al. Moscow: Institute of Applied Mathematics, USSR Academy of Sciences, 1979, 23 pp.

65. Bulysheva, L. A., Chernysheva, R. T. "Organization of ISTOK System Program Packages From Modules In Various Algorithmic Languages." Voprosy razrabotki i ekspluatatsii paketov prikladnykh programm (Problems of Development and Use of Application Program Packages). Novosibirsk: ITPM, Siberian Division, USSR Academy of Sciences, 1981, pp. 101–131.

66. "The Experience of Using Modular Technology to Solve Problems of Flow Around a Body." V. F. Volkov, S. B. Zhibinov, B. P. Kolobov, et. al. Zadachi aerodinamiki etel prostranstvennoy konfiguratsii (Problems In the Aerodynamics of Three-Dimensional Bodies). Novosibirsk: ITPM, Siberian Division, USSR Academy of Sciences, 1982, pp. 20–58.

UDC 681.31-192

Software Methods and Facilities For Increasing Fault Tolerance of Control Computer Systems

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian (manuscript received 21 Jul 86) No 3, May-Jun 1988, pp 30-36

[Article by I. V. Shturts, A. B. Romanovskiy and V. R. Vasilev]

[Text] **Introduction.** The area of application of control computer systems is constantly expanding. The number of important applications requiring high computer system reliability is also increasing, where failures might lead to great material losses or even endanger human health and life. This is characteristic of certain process control systems and on-board control systems. If we take the term computer system to include a hardware-software complex consisting of a computer, operating system and application software, the sources of failures may include defects and errors in any component of the computer system, as well as erroneous actions by the human operator for distortion of transmitted data. The reliability level of general purpose control computer systems is insufficient for important applications unless special steps are taken to increase reliability. These steps can be of two types: Prevention of failures and provision of fault tolerance[1, 2].

Steps of the first type are intended to eliminate errors in the design of computer systems, to increase the reliability and durability of system elements. These steps include comprehensive testing of computer hardware, reducing the loads on electronic elements, as well as steps to eliminate program errors: Careful planning, verification, testing and debugging of programs. It is impossible to achieve absolute reliability of hardware elements (due to physical aging) or complete freedom from program errors (due to their great complexity); therefore, steps of the second type are very important. They are intended to give computer systems the property of fault tolerance, i.e., the ability to detect faults and eliminate or reduce their harmful effects and continue correct functioning.

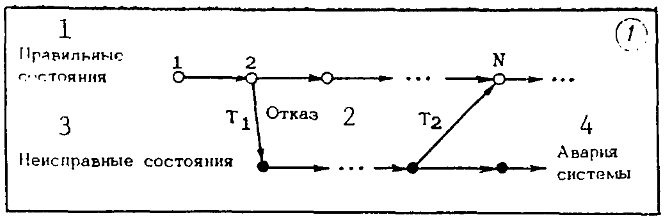
These steps include, for example, redundancy (duplication, voting) of computer hardware modules, long used as a method of decreasing hardware failures. Since many of the functions of modern control computer systems are implemented in software rather than hardware, it is important to extend these measures in order to neutralize software errors, a technique not yet sufficiently used.

This article is dedicated to this second trend. Fault tolerance is improved by the use of special programs which expand the capabilities of the operating system and are available for use by application software. It is these programs, their design principles and algorithms, which are the basic subject of our research.

Faults and errors. The general engineering concept of a fault as a loss of serviceability by an object must be refined in its application to control computer systems in which most of the functions are implemented not in hardware, but in software. At some level of abstraction it is possible to find common points in hardware and software faults, which are usually contrasted. The basis for this abstraction is to be found in the concepts of system and process.

A computer system is frequently represented as a hierarchy of hardware and software systems, each of which is a combination of interacting elements in systems at lower levels of the hierarchy. The interaction of these elements is determined by the structure of the system, which is embodied in the plan (design) of the computer system.

A defect (error) in any system making up the computer system appears as an intolerable deviation of its behavior from the required functional specifications. A fault is an event causing a defect. The vulnerable point of this definition is that that specifications of a computer system and its higher level subsystems are usually incomplete, contradictory and poorly documented. This results from the difficulty of generating precise and complete specifications of complex systems, a fundamental problem of computer science.



State Diagram of Functioning Process of a System With a Possible Fault. 1, Correct states; 2, fault; 3, incorrect states; 4, system emergency.

The functioning process of each system is considered a sequence of transitions from one state to another. Faults are transitions leading to erroneous (or defective) states; from these states the system may enter a state in which a defect is manifested[2] (Figure 1). A defective state is referred to as a state with an undetected fault. For example, a fault causing incorrect writing of data on a medium shifts the peripheral storage system to a defective state; the defect may be discovered later, when the error appears upon reading or use of the distorted data.

The reason for the transition of the system to a defective state is always an internal system fault. Two types of such faults are possible: Faults in elements and structural faults. A fault in an element is caused by a defective state of the element. For example, breakdown of diode, distortion of a block of data as it is written to disk. Structural faults are caused by improper interaction of correctly operating elements, the reason most frequently being an error in planning or a system design defect (these are called design faults in hardware). For example, improper connection of electronic elements by wires; a disagreement between the actual parameters in a subroutine call operator and the formal parameters in its description. A transition to this type of defective state usually occurs in the planning or manufacturing stage; exceptions include cases in which contacts fail during hardware use or programs stored in computer system memory are distorted.

The differences between these two types of internal faults are significant from the standpoint of fault tolerance: Hardware faults are easier to detect and correct than structural faults. Hardware faults are usually of the first kind, software faults of the second. This is because in complex systems the hardware is much simpler than the software, having a smaller number of component elements, interconnections and states, so that hardware planning errors are rather rare events.

The difficulties which we have mentioned in specifying the design of complex software mean that software unavoidably contains errors not revealed during testing and debugging, making the reliability level of software lower than that of hardware[3]. We note, however, that this difference is reduced as the degree of integration of hardware elements increases in LSI and VLSI devices: The complexity and varieties of interconnections among elements on a chip make full testing difficult and increase the probability of a plan fault. On the other hand, hardware is subject to physical wear and aging, while software is not; therefore, the instructions of a program, considered as elements, are always "serviceable." We note that here also the difference is relative rather than absolute: As the requirements placed on a computer system change, programs are modified and new errors are introduced, the appearance of which is similar to aging and deterioration of program characteristics.

Basic principles of fault tolerance. The basic principle used to provide fault tolerance in technical systems is resource redundancy. In computer systems, hardware (structural), time, data and software resources are made redundant in various combinations. We shall not further consider structural redundancy, since it is little used in standard general purpose computer configurations.

Efforts toward assuring fault tolerance can be subdivided into two stages: Detection of defective states and recovery of correct states. The transition from the first stage to the second occurs in response to an exception signal when an error of some type is detected; the computer system begins to execute an exception handler program which contains the algorithm for recovery.

Fault-tolerant facilities must meet the following requirements:

— Defect isolation (they should not propagate through the computer system, for example by copying distorted data). This may occur when any subsystem is in a defective state. Defect isolation is facilitated by special computer system structuring techniques, as well as by meeting the next requirement;

— earliest possible detection of defects without detailed fault diagnosis;

— the use of a software process (task in the sense of the operating system) of protection from faults with various types of redundancy (in addition to structural redundancy at the hardware module level). First of all, this agrees well with the modern practice of specification and planning of control computer system software as a system of parallel processes; secondly, we have defined the concept of the defective state and transition specifically for processes.

Testing and restoration methods. There are various methods for testing, i.e., detection of defective states of an element (system), distinguished by the method of establishing the fact of a fault. Let us list the most common ones.

Testing by copying: A copy of a process is executed independently of the original, and equality of the results is verified; recomputation with comparison of results, storage of duplicate data structures, etc., similar to duplication and voting among hardware modules. This type of testing, quite costly in terms of redundancy, does not detect structural failures, but is quite effective for detecting occurring faults, i.e., failures in elements.

Testing by coding: Based on redundancy in the representation of data, with additional bits used to reveal data distortion. For example, parity checking as bytes are transmitted, checksums for data blocks. Both software and hardware failures are detected. The method is economical.

Time testing: Limitation of the time allotted for execution of individual process phases. Plan faults and faults in elements causing excessively slow operation are detected. The method is important for real-time systems.

Acceptability testing: Verification that the state of a process corresponds to the specifications of acceptable behavior provided by the developer. This specification may be represented as statements concerning the values of variables at certain points during the process; for example, file index within certain limits, value of an angle (less than 180°), etc. This method is widely used to reveal software errors both during debugging and during use of computer systems; data distortions are also detected.

Diagnostic testing: Testing of the correct operation of element by providing test input signals, for which the correct output data are known. This method is commonly employed to test computer hardware.

From the standpoint of program implementation in a standard operating environment, testing methods are usually divided into three types: Forced, active and passive. Forced testing is the primary reaction to emergency signals generated by computer testing equipment or operating system programs (memory protection, overflow, etc.); passive testing implements time and acceptability tests; active testing refers to diagnostic tests.

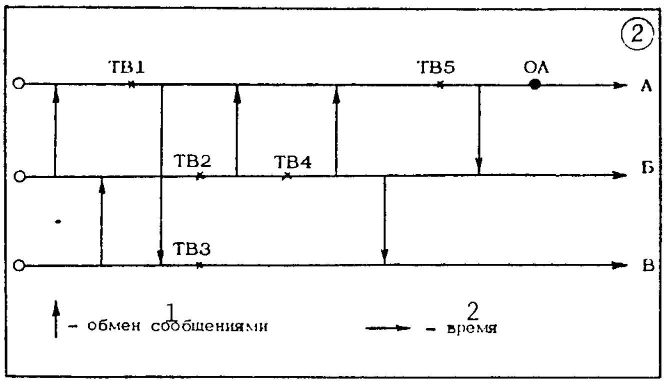
Recovery may be achieved by rollback — returning the process to the correct state preceding occurrence of a fault, or by correction — transition of the process to the correct state without returning to a previous state. In Figure 1, rollback is illustrated as a recovery from any defective state, for example to state 2, while correction is represented as the transition T_2 to state *N*. In order for rollback to be possible, certain states of the process, potential rollback points, called recovery points or check points (in the terminology of the OS YeS operating system) must be recorded. We note that in this cases there is no guarantee that the recovery point will be a member of the set of correct states. Rollback utilizes a significant portion of the time reserve, since in order to return to normal operation the process must repeat a sequence of states already passed. Obviously, rollback is an effective means of dealing with hardware faults.

Correction is much less universal than rollback, since it requires knowledge of the specifics of the process in order to neutralize defects.

Other possible recovery methods include: Masking of faults by voting when copy testing is used; reconfiguration (for example execution of an alternative version of a program, possibly less precise — with degradation) and soft shutdown.

Rollback of interacting processes. The problems of isolating defects and selecting recovery strategies are most complex when processes which exchange synchronization signals and/or messages must be rolled back. When a fault arises in one such process, not only must it be rolled back, but coordinated rollback of other, interacting processes is required. Otherwise, synchronization of processes may be lost or distorted messages propagated. Coordinated rollback of several processes requires that their number and the extent of the rollback both be minimized, particularly for real-time systems. A clear example of the impossibility of this requirement in some cases is the domino effect[2, 4], in which the rollback "wave" expands uncontrollably in width (in the number of processes involved) as well as depth (time).

Figure 2 shows conditions under which the domino effect may arise for three processes. Obviously, after discovery of a fault in process *A* at moment *OA*, all three processes must be returned to their initial states, and some of the recovery points TB (TB1–TB5) cannot be used. The problem thus arises of developing coordinated rollback algorithms for interacting processes.



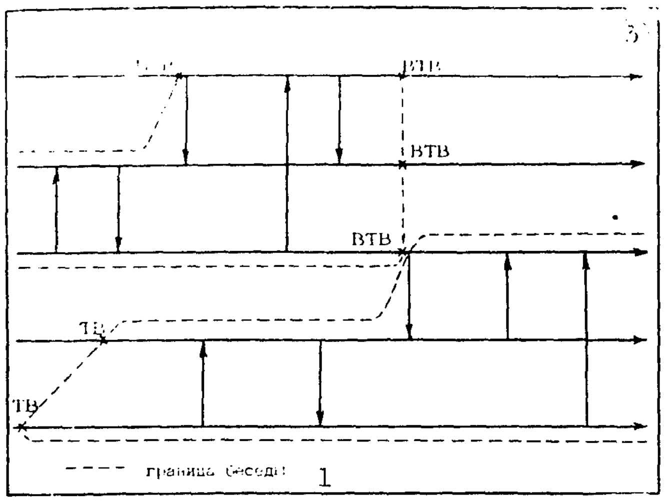
The Domino Effect For Three Interacting Processes. 1, Exchange of messages; 2, time; TB — recovery point..

A number of algorithms of this type have been suggested. Some of them include no actions to prevent the domino effect, but it is detected[2, 5]. When these algorithms are executed, a dynamic list of related processes and recovery points within them is generated, and in case of a failure in a process, this list is used to roll back the required processes. Other algorithms[4, 6] require development of all application programs in parallel programming languages which include special structures describing coordinated actions to be taken when rollback is required in various processes: Establishment of recovery points, testing of correct functioning, rollback, exchange of information. In addition to the fact that this requires the development of new programming languages, the use of such algorithms is not acceptable from the technological standpoint due to the impossibility of independent development of processes.

Let us analyze a rollback algorithm which does not have these shortcomings. Its essential advantage is that it can run under both universal and specialized operating systems, in strongly and weakly coupled systems, and on networks. The algorithm is implemented in a rollback subsystem, a special software layer between the operating system and the processes.

The algorithm is based on the concept of the conference — a dynamically created set of interacting processes. For each conference there is a conference list — a data structure describing a set of pairs {PROCESS ____ NAME, TB}. If one process fails, all processes in the conference are rolled back to a TB. Each process can be included in only one conference; inclusion in a conference occurs before any interaction. At each moment, there may be several conferences in existence in the computer system. Conferences are limited in size, i.e., the number of participants and/or time of existence; limitations are selected on the basis of the time reserves and planned computer system throughput characteristics.

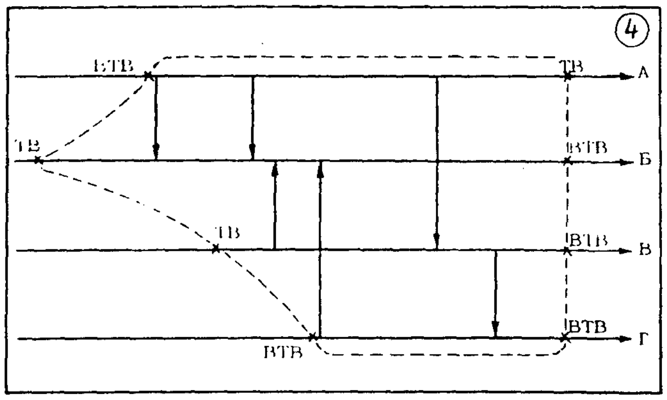
The essence of the algorithm consists of testing the size and length of conferences and establishing, when necessary, additional recovery points (BTB) in addition to the recovery points established at moments planned by the process designer. These initial recovery points are inserted to assure correct completion of the conference, which occurs in one of two cases:



Example of Breakdown of a Conference Upon Reaching the Maximum Size. 1, conference limit.

— A request for interaction with a process not included in the conference occurs, while the size of the conference has already reached the maximum permitted (Figure 3 — size of conference limited to three processes);

— one of the processes must establish a planned recovery point (Figure 4).



Example of Breakdown of a Conference Due to Setting of Planned Recovery Point In Process A.

It can be demonstrated that synchronous departure of processes from a conference with simultaneous setting of supplementary recovery points guarantees that the domino effect will not occur.

The rollback subsystem processes information on sending of messages, setting of recovery points and faults. Processes access the subsystem through the following instructions.

SEND (sending process name, receiving process name)

The rollback subsystem in this case performs any actions needed to reformat the lists of conferences, establishes supplementary recovery points in processes, then allows the information to be sent.

FAULT (process name, fault detected)

This subsystem rolls back all processes in the conference to the moment when they entered it.

SET_TB (process name)

The subsystem reformats the conference lists as necessary and sets a supplementary recovery point.

Thus, the basic actions performed by the subsystem with respect to processes are as follows:

ROLLBACK_TO_TB (list of processes in conference)
SET_BTB (process name).

These instructions and actions allow any rollback algorithm[2, 4-6] to be implemented. The rollback subsystem may be centralized or decentralized. If centralized, it is a special program storing information on conference lists and performing all actions necessary to roll back interacting processes. In the latter case, implementation of the subsystem is more complex. It includes the corresponding protocols to "take control" of a process, perform rollback, as well as protocols to maintain identity of information on conferences stored in various processes. This implementation is the only possible one for distributed computer systems.

The use in the rollback subsystem of an algorithm based on limiting the size and time of existence of a conference simplifies recovery of a real-time system with high rollback cost and limited time reserve. Let us reiterate the main advantages of this algorithm: Elimination of the domino effect; reduction in rollback depth (with a guaranteed maximum); simplicity of determination (after discovery of a fault) of processes which are interacting and recovery points for the rollback; reduced number of recovery points; automation of all actions related to recovery, which become transparent to the user; reduced number of processes rolled back.

The "Phoenix" software package to make the OS RV real-time operating system fault tolerant. The most common multitasking operating system for SM-4 control computers, capable of operating in real time mode, is OS RV. It is this operating system which is recommended for use at the upper levels of automated process control systems and ASU GPS.

OS RV contains an extensive set of test facilities (tests for hardware devices, capture of synchronous system interrupts from tasks, stack and memory protection, error recording facilities), but their use for fault tolerance purposes is difficult, requiring a high level of programmer qualifications, or quite impossible. Thus, the need has arisen to develop supplementary software facilities to facilitate and partially automate the development and implementation of fault-tolerant real-time control systems. These facilities might be provided in the form of a software package, a superstructure above OS RV, with the following properties:

— Ability to utilize standard OS RV test facilities;

— provision of supplementary test facilities for both hardware and software;

— provision of facilities to describe recovery algorithms for both individual tasks and the entire computer system;

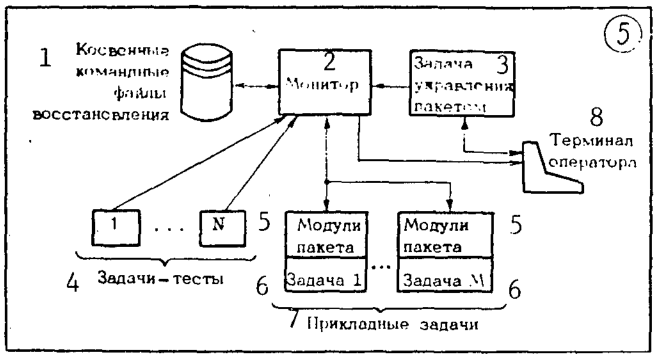
— ability to add new testing and recovery capabilities easily;

— facilities for adjustment to specific control computer configurations;

— provision of these capabilities to application programmers.

Such a software package has been written for OS RV and the DOS operating system of the M6000 control computer[7]. In contrast to this existing package, the "Phoenix" package is designed for the universal OS RV operating system of the SM-4 control computer, and provides great automatic computing process recovery capabilities in real time.

The "Phoenix" package contains programs which monitor the computer system and support its normal functioning when hardware and software faults occur. The users of the package are application programmers, who use the facilities of the package as they develop application programs, as well as system programmers, who adjust the parameters of the package to specific control computer configurations. These users can use both the standard package facilities and their own facilities, based on the common mechanism to signal faults and start the recovery process.



Structural Diagram of the "Phoenix" Package. 1, In directory cover command files; 2, monitor; 3, package control task; 4, test tasks; 5, package modules; 6, task; 7, application tasks; 8, operator's terminal.

The forced testing facilities of the "Phoenix" package are special modules added to application programs which detect abnormal termination of tasks and start the recovery process.

Active testing is performed by modules attached to application programs, as well as special tests functioning as independent tasks in background mode. The first version of the package includes three testing tasks: A processor instruction test, interrupt test and operating system pool test, plus a module to test the memory used by a task.

The package contains linguistic facilities for passive testing, allowing application programmers to formulate statements on the status of variables as logic expressions, on the maximum permissible time for execution of a program fragment or entire program. In the latter case the status of a task is monitored by facilities external to the task, increasing the reliability of the monitoring. Formulating these statements, programmers can limit the execution of loops, check variables at moments when empirical limitations on their values can be stated, monitor the correctness of all input and output data. Thus, these can be considered dynamic program specification test facilities.

An example of a statement in a macroassembler program:

```
ASSERT LT, INDEX, #20
AND GE, INDEX
HANDL #3
```

It is asserted that the variable INDEX is less than 20 and greater than or equal to 0. If this condition is not met, recovery handler number three is to be used.

The "Phoenix" package uses the rollback strategy to restore normal computer system functioning. If a fault is detected, the package automatically calls the recovery handler appropriate for the fault. Some of these actions are described by application programmers in the development language and/or the language of OS RV indirect command files. In addition to the mechanism to start recovery handlers, the package also contains programs to set recovery points and roll back to them. A recovery point includes files used by the application programmer. The following is an example of description of recovery actions in the OS RV indirect command file language:

```
REM TASK; REMOVE TASK
PIP/NV=DK2:TASK.TSK; RECOVER PATTERN
                   ; RESTORE TASK IMAGE FROM RESERVE COPY
INS TASK; INSTALL TASK FROM RESERVE COPY
RUN TASK; RESTART TASK
```

If a recovery point was set, the task is restarted from it.

The "Phoenix" package also implements a centralized coordinated rollback strategy for interacting processes based on a system of conferences with limited numbers of participants. The structure of the package is shown in Figure 5.

Using the "Phoenix" package regulates the actions of programmers providing fault tolerance during development of application programs, requiring that they use a standardized method of providing fault tolerance. The package also includes a debugging operating mode and can be of significant assistance in debugging programs.

The package has an open structure and can easily be expanded with new testing and recovery facilities. In particular, for use in distributed computer systems a few test tasks (equal to the number of nodes) must be developed and installed in the various network nodes to exchange verification messages[2].

The package is written in assembler and can be used during development of programs in FORTRAN, PASCAL or macroassembler under OS RV. In operation, the package occupies about 10 K of main memory and requires a small percentage of processor time.

Experimental operation of the package has demonstrated that it neutralizes most faults characteristic of OS RV on the SM-4 control computer.

REFERENCES

1. Lipayev, V. V. Nadezhnost programmnogo obespecheniya ASU (Automatic Control System Software Reliability). Moscow, Energoizdat Press, 1981, 240 pp.

2. Anderson, T., Lee, P. A. "Fault Tolerance. Principles and Practice." New Jersey, Prentice Hall, 1981, 369 pp.

3. Longbottom, R. Nadezhnost vychislitelnykh sistem (Computer System Reliability). Moscow, Energoatomizdat Press, 1985, 288 pp.

4. Randell, B. "System Structure For Software Fault Tolerance." IEEE Trans., 1975, Se–1, No. 1, pp 220–232.

5. Wood, W. G. "A Decentralized Recovery Control Protocol." FTCS–11, 1981, pp 159–164.

6. Antola, A., Skarabottolo, N. "Retrospective Error Elimination In Distributed Systems." Avtomatika i vychisl. tekhnika, 1986, No. 2, pp 25-32.

7. Timofeyev, B. B., Ushakov, V. A., Kostenko, V. S. "Functional Capabilities of Operational Testing and Recovery Software Package For Real–Time Systems." USiM, 1978, No. 1, pp 17-25.

UDC 681.3

One Method of Individual Adaptation of OS YeS Operating System To User Requirements

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian (manuscript received 18 Aug 86) No 3, May-Jun 1988, pp 36-38

[Article by Ye. P. Lilitko]

[Text] The creation of flexible and powerful software systems on the YeS computers, satisfying large numbers of users with various levels of training and different requirements, as well as better adaptation of the operating system (OS) to the requirements of a specific user, demands facilities allowing the OS and its components to be adjusted without making changes to its modules or influencing the operations of other users.

An interesting method for replacement of access method programs was suggested in[1]. This method allows servicing of nonstandard input-output devices with standard facilities (for example, using the standard printer output facilities provided by any programming language to organize output to a terminal or any other device); it is also possible to examine the output file of a task at a terminal before the task is completed, sometimes quite useful for debugging of programs with long computation times. An example was presented in[1] of the use of the method to organize a library of procedures, and this list of applications can be continued. However, as the authors noted quite correctly, the method they suggested does have a significant shortcoming, in that it cannot be used to replace resident access method modules of OS ES (due to the peculiarities of the IDENTIFY macroinstruction), nor for OS YeS in SVS mode (in this mode all access method modules are resident).

A method was suggested in[2] for OS YeS program interception, based on preprocessing of interrupts accessing the supervisor (*SVC* interrupts) by a preprocessing program. This method eliminates the limitations of the method of[1] and provides a somewhat broader list of capabilities. However, it also has defects. First of all, the preprocessing program must process all *SVC* interrupts, not just interrupts with codes 6, 7, 8 and 42 (the macroinstructions LINK, XCTL, LOAD, ATTACH) and although processing of the "other" interrupts requires no more than 8-10 instructions, considering that the number of *SVC* interrupts is quite high, the operation of the OS is noticeably slower. Secondly, many OS YeS measurement monitors, such as[3], also use the idea of a preprocessor, and simultaneous operation of two preprocessors in the OS is undesirable, since it doubles the time loss, and also the slightest inconsistency may cause undesirable effects in the operation of the OS.

This article suggests a method of individual OS adaptation combining the simplicity and correctness of the first of these methods and the universality of the second. The method suggested can also be used and is presently being used in a number of cases, for example:

— To organize visual observation of the course of a computing process during debugging of complex systems;

— to organize coprograms and data exchange through "internal" files;

— for operations in higher-level languages with input-output devices not provided for.

— to construct convenient processing procedures based on OS YeS utilities.

The idea of the method suggested is quite similar to that of[1], but some of the peculiarities of the open data set (OPEN) and load module into memory (LOAD) macroinstructions are used to avoid the limitations inherent in[1]. The algorithm for interaction of the OPEN and LOAD macroinstructions when a data set is opened is rather complex, but for our purposes the simplified plan outlined below is sufficient.

The OPEN macroinstruction:

— The access method and usage mode are determined from the data control block (DCB);

— the input-output program of the access method is loaded by means of the LOAD macroinstruction;

— the address of the input-output program is placed in the DCB of the data set being opened.

The LOAD macroinstruction:

— Checks for the presence of the load module in the area assigned to the task (this fact was used in[1]), in the task library (task point), in the OS YeS common area (LPA), in libraries in the list contained in the LNKLST00 segment of the SYS1.PARMLIB library;

— loads the module when found into memory or generates an error message if the module is not found.

In the task area or OS YeS common area, a physical load of the module does not occur, but rather the access counter of the module is incremented.

We can see from this module search algorithm that if a program with the name of the access method load module is placed in the task library, when the LOAD macroinstruction is executed this program will be loaded rather than the resident module from the LPA. Furthermore, using this method we avoid the IDENTIFY macroinstruction and, consequently, eliminate all the limitations of method[1].

Thus, to replace an access method program with another, we need but place the replacement program in the personal load module library under the name of the system module to be replaced and, when the user program starts execution, indicate the personal library in a *DD* statement with the name STEPLIB. When the OPEN macroinstruction is executed, the program from the personal library will be loaded in place of the system access method module. However, we must recall that the access method imitation program must be reentrant (if the user program does not utilize the capabilities for asynchronous execution of input-output operations, the access method imitation program need not be reentrant, but it must be flagged as such during linkage editing).

We would also like to note some aspects of the implementation of the method, resulting from the specifics of access method module functioning. In practice, it is most frequently necessary to replace BSAM and QSAM sequential access method modules, and therefore we shall discuss them in more detail.

When an access method is replaced, as was quite properly noted in[1], it makes sense to insert the parameter DUMMY in a *DD* statement, to avoid distribution of memory in the working volume for the data set and reduce task initiation and/or completion times. In this case, the module IGG019AV should be replaced, i.e., the access method imitation program must be written and placed in the task library (task point) under the name IGG019AV. The parameters passed to the access method module in the general purpose registers usually depend on the operating mode of the PUT (GET) macroinstruction. They can be examined by printing out the PUT (GET) macroexpansion by setting the GEN parameter in the PRINT operator. However, and this is quite important, there is another problem which was unfortunately not discussed in[1]. The IGG019AV module has a nonstandard structure for OS YeS and looks like:

```
IGG019AV  CSECT
          DC        A( )   PUT program address
          DC        A( )   PUTL program address
          DC        A( )   GET program address
          DC        A( )   CHECK program address
          DC        A( )   IOB address
          DC        A( )   BSAM entry address
          DC        A( )   CNTRL entry address
          .
          .
          .
PUT      EQU        *
          .
          .
          .
         END        IGG019AV
```

Obviously the access method imitation program must also have the same structure, otherwise the OPEN macroinstruction will not work correctly.

There is yet another peculiarity relating to both of the access methods in question. If the access method imitation program itself does not perform "real" input-output operations (for example, when coprograms are organized), no difficulties arise; otherwise, one must be concerned with opening data sets and requesting memory allocation. Of course, the access method imitation program may contain a flag and upon first entry open sets and allocate memory, skipping this operation upon later entry, but this requires additional testing each time the access method imitation program is entered, which decreases the effectiveness of the program. It is better to write a program monitor which will:

— Receive in the PARM field of the EXEC operator the name of the user program and PARM field information for it;

— load the access method imitation program;

— open the data sets required for operation of the access method imitation program and request the buffer pool and operating areas;

— place the addresses of the requested areas and data control blocks (DCB) of the open sets in a memory area available to the access method imitation program (possibly within its body);

— start the user program with the rights of a monitor subtask;

— when the user program terminates, close the data sets and release the memory allocated;

— use the EXTRACT macroinstruction to determine the user program completion code;

— terminate operations with the same code as the user program (either by placing it in register 15, or by means of the ABEND macroinstruction).

This approach not only avoids testing each time an input-output operation is performed, but also solves the problem of closing the data set and releasing memory after all operations are completed.

This method is particularly convenient when writing dialog programs in high-level languages, since a printed line is handed to the access method imitation program after it is fully formatted and the user need not be concerned with data conversion, since this is performed by the standard output formatting facilities of the language.

The author has used this method many times to write dialog programs and catalogue procedures, to organize interaction with ES/7920 stations, etc.

REFERENCES

1. Gusev, V. V., Petrukhina, A. V. "Intercepting Access Method Programs in OS YeS." USiM, 1984, No. 1, pp 65-68.

2. Zinchenko, A. N., Kritskiy, S. V. "Organizing a Debugging Environment By Intercepting OS YeS Programs." ibid, 1985, No. 6, pp 87-89.

3. "The ISDSLEDOVATEL (RESEARCHER) Automated System To Study the Effectiveness of Application Program Packages. Composition, Functions, Capabilities." I. V. Sergiyenko, V. G. Voronenko, S. Yu. Lysenko and V. V. Skopetskiy. ibid, 1985, No. 5, pp 96-99.

4. Danilochkin, V. P., Odintsov, B. V., Peledov, G. V. Spravochnik sistemonogo programmista po operatsionnoy sisteme OS YeS (OS YeS Operating System Programmer's Handbook). Moscow, Finansy i statistika Press, 1982, 288 pp.

UDC 681.3.06.066

Dynamic Terminal Redistribution In Os YeS

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian (manuscript received 05 Aug 86; after revision 17 Feb 87) No 3, May–Jun 1988, pp. 39–40

[Article by V. A. Akulov, Yu. Ye. Klepikov and V. A. Kozlov]

[Text] Users of the Os YeS operating system (MVT and SVS mode) have at their disposal many different interactive systems, including PRIMUS, JEC, DUVZ, etc. The problem of dynamic redistribution of terminals among these systems, i.e., "disconnection" of a terminal from one interactive system and its "connection" to another, with a possible later return of the disconnected terminal, is a typical problem at many computer centers. Also there are typically terminals not presently in use, but which cannot be transferred to another interactive system, since they have already been distributed.

This redistribution is usually performed when the interactive systems stop active operation. Each such reloading causes nonproductive expenditure of machine time and results in user downtime of 5–7 minutes.

Resources (main memory, peripheral devices, data sets) are distributed in OS YeS in the task initialization stage. The resources thus distributed become unavailable for use by other tasks until the task to which they are assigned completes running.

Whereas this type of resource distribution is suitable for most tasks, it creates serious inconveniences for terminal utilization, since dynamic redistribution among tasks in simultaneous operation is impossible.

The basic information on the functional purpose, technical characteristics, capabilities and current status of devices, including terminals, is contained in the system block UCB (device control block).

One of the simplest methods of dynamic redistribution of terminals is to change the contents of the UCB[1, 2]. If certain conditions are met, this method is effective for the OS[2]. Based on this method, the authors have developed a "dynamic peripheral device reconfiguration monitor," which meets the following requirements:

— Independence with respect to active interactive systems (in contrast to[3]);

— ability to be present in memory only at the moment when system blocks are modified (in contrast to[2]) and to complete operation independently of the interactive systems;

— centralized accounting and storage of data on terminal status;

— ability to manipulate not only individual terminals, but also groups of terminals in a single operating cycle (in contrast to[2]).

The monitor is designed for dynamic redistribution of local YeS7066, YeS7927 and YeS7063 terminals in YeS7920 mode among interactive systems in the OS YeS environment. Operation of the monitor is controlled in dialog mode by entering monitor instructions at the computer operator's console. The monitor recognizes the following instructions:

O — logically disconnect a terminal or group of terminals from an indicated interactive system and shift them to on-line status;

F — release a device or group of devices with indicated addresses, shifting them to on-line status;

R — logically reconnect a terminal or group of terminals, previously disconnected from an interactive system by the O instruction, restoring their status at the moment they were disconnected;

S — logically switch a terminal or group of terminals from one interactive system to another;

P — delete previously stored information on the status of terminals for one or several interactive systems from the list;

L — send data on the current status of terminals for one or more interactive systems to the operator's console;

H — send information on the composition and structure of the monitor instructions;

EOJ — end monitor operation.

When terminals are dynamically redistributed, information on their current status is preserved in a special memory area, the "reference list." When the monitor is loaded, this list is kept in main memory, when the monitor finishes operations the list is copied to peripheral storage (as a data set on magnetic disk). Normal functioning of the monitor requires 30 kbytes of RAM.

The monitor dynamically redistributes terminals only among interactive systems using standard access methods (GAM, BTAM, TCAM), supporting the system control block structure used in OS YeS (TCB, TIOT, UCB) and standard interactions among them.

The process of starting an interactive system is as follows. First the basic interactive system IS_1 is started on all terminals. Some of the terminals are released by the monitor instruction O, and interactive system IS_2 is started in these terminals. Subsequently, these terminals can be used for IS_1 and IS_2. A monitor S instruction is used to switch a terminal (or terminals) between the systems. In case of an emergency termination of one of the interactive systems, the users of the other interactive systems continue to operate, since the integrity of the operating system is not disrupted. This method can also be used for larger numbers of interactive systems.

The functions of the computer operator using the monitor are as follows. The computer operator starts and controls operation of the monitor by entering instructions. Since the operator can always output the monitor command structure at the console (by means of the H instruction), and the correctness of his actions (responses) is always rigidly checked by the monitor, its use presents no difficulty to the operator and does not increase his workload.

The results of using the monitor at several computer centers under OS YeS (version 6.1, MVT and SVS modes) have demonstrated that its capacity for dynamic redistribution of terminals is sufficient for most instances. The monitor can reduce nonproductive machine time use resulting from reloading interactive systems in order to redistribute terminals. Furthermore, the monitor increases the "accessibility" of terminals by switching them among interactive systems.

The "dynamic peripheral device reconfiguration monitor" program was deposited with OFAP in 1985 (No. 4269–OFAP).

REFERENCES

1. Dudnik, V. A. "Terminal Redistribution Methods Under OS YeS." Programmirovaniye, 1985, No. 2, pp. 39–43.

2. Bezrukov, N. N., Pribytkin, A. Ya. "Dynamic Terminal Redistribution Dispatcher for OS YeS." ibid, 1986, No. 6, pp. 62–63.

3. Deykalo, G. F. Novye sredstva programmirovaniya dlya YeS EVM (New Programming Facilities for YeS Computers). Moscow, Finansy i statistika Press, 1984, 160 pp.

4. Spravochnik sistemnogo programmista po operatsionnoy sisteme OS YeS (OS YeS Operating System Programmer's Handbook). Edited by L. D. Raykov. Moscow, Finansy i statistika Press, 1981, 96
pp.

UDC 681.3.012

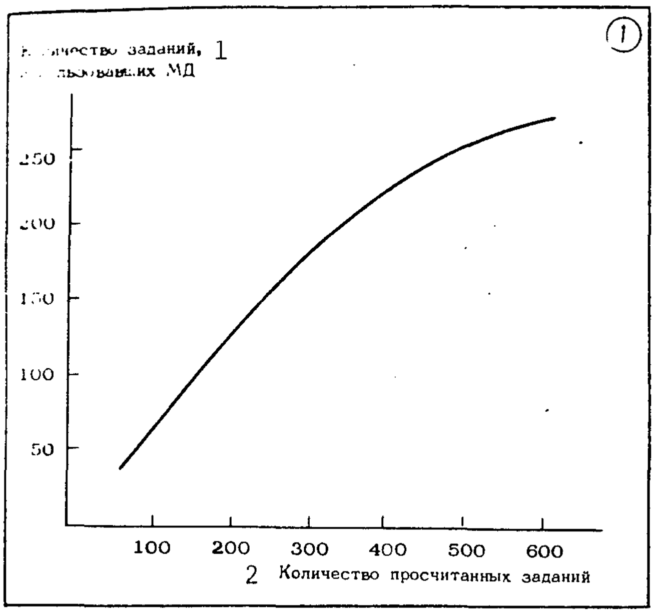
Organization of VTsKP Automated User Archive

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian (manuscript received 22 Apr 86; after revision 17 Dec 87) No 3, May–Jun 1988, pp. 40–42

[Article by G. S. Serykov, A. G. Zyryanskiy and A. D. Savchenko]

[Text] The increase in volumes of information processed and the number of users at the computer center of the Institute of Cybernetics, Ukrainian Academy of Sciences, has increased the need to organize and maintain an automated user archive as an independent level in the hierarchical memory system of the computer system.

In spite of the significant number of disks which can be accessed at the computer center, the demands for disk storage have not decreased. The large number of disks requires expenditure of nonproductive time in exchanging disk packs during execution of remotely and locally loaded application packages. For example, the number of disks used by software packages each day has risen greatly (Figure 1) and resulted in a reduction in multitasking and a decrease in computer system throughput in general. A study of the magnetic disks used at the computer center has shown that a high percentage of all data sets consists of library and sequential files, 80% of which could be stored in a shared automated archive.



Use of Magnetic Disk by Batch Tasks. 1, Number of tasks using magnetic disks; 2, number of tasks read.

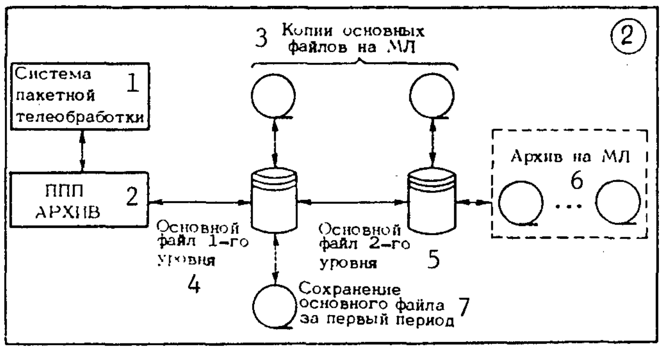
To support effective access to user source and object modules both in the interactive debugging mode and for "quick package" operation without interchanging magnetic disks, the computer center has developed a technology and supporting software to organize the ARKHIV automated application software package archive using the library service system.

This system is designed to store and retrieve information and supports effective management of user data. One component of the library service system is its main file — a distributed storage area on disk (or tape), where symbolic text is recorded in compressed form. Each main file consists of one or more modules. A module is a set of data records and their related control information.

A main file on disk is a set of direct access method data consisting of blocked fixed-length records. The number of main files, block size and number of blocks are determined as each main file is planned.

A main file on tape is a sequential data set located on magnetic tape. All data stored in a main file are located on tape and copied from one tape to another as they are processed. Modules are organized on each tape in increasing alphanumeric order by module name.

Three levels were defined as the archive was planned (Figure 2): The first two levels are the operational archive, while the third level is a long-term archive on magnetic tape. The main factor considered in determining the location of user data at the proper level is the data access frequency. The first level operational archive contains modules with which the user works actively over a predetermined time period (for example, one week).



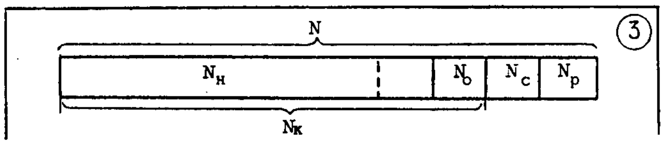
2
3 Копии основных файлов на МЛ
Система пакетной телеобработки 1
ППП АРХИВ 2
Основной файл 1-го уровня 4
Основной файл 2-го уровня 5
Архив на МЛ 6
Сохранение основного файла за первый период 7

Functional Structure of Archive. 1, Remote batch processing system; 2, ARKHIV application software package; 3, copies of main files on magnetic tape; 4, first level main file; 5, second level main file; 6, magnetic tape archive; 7, main file backup for first period.

At the end of the first period, the contents of the main file (OF1) and user activity are determined. These data are then used to make a decision concerning transfer of some of the modules to the second operational archive level (OF2).

At the end of the second time period (for example, one month), the contents of the second operational archive level and user activity working with data at the second level are determined, and used to make the decision to transfer some of the modules to the third level archive on magnetic tape for long-term storage.

At the end of the first and second periods, the volume of memory which must be released (N_0) in main file OF1 or OF2 is determined, considering its filling at the end of the current period, the intensity of user interaction, and the length of the plan period (Figure 3).

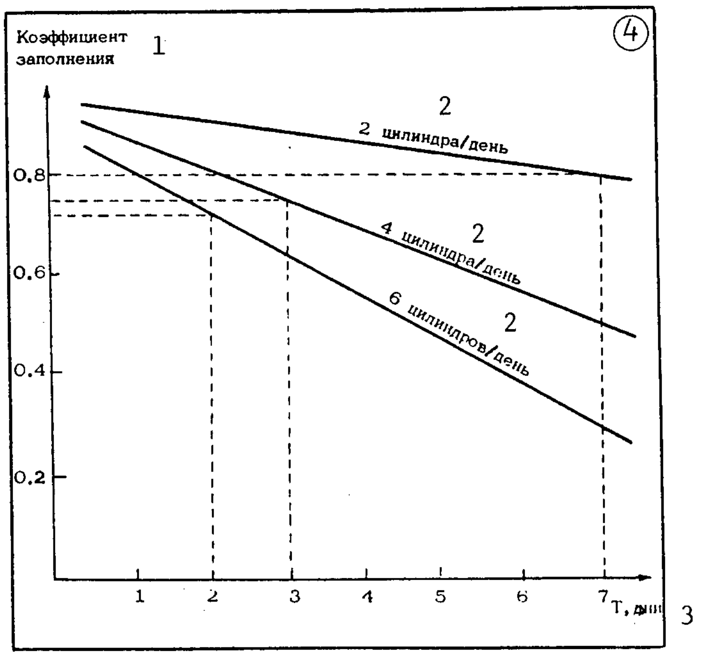


N - Main file volume (number of tracks)
N_H - Main file contents at beginning of current period
N_K - Main file contents at end of current period
N_0 - Number of tracks to be released
N_C - Number of tracks unused in current period
N_p - Memory space reserved

Schematic Diagram of Main File Structure.

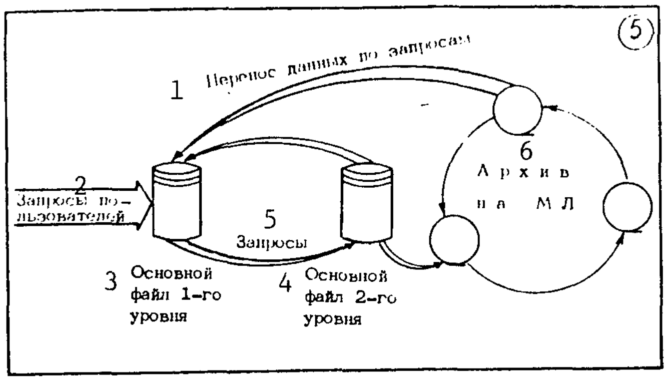
Figure 4 shows the variation in effective main file use as a function of its filling rate by users and the length of the filling period. As the filling intensity increases, the length of the period correspondingly decreases. The dotted lines show effective selection of a period length for each example. The main file filling time depends on the size of the main file, the number of users and their main file usage intensity. The intensity of usage of the main file is not constant; in order to smooth fluctuations and remain within the boundaries of the main file, a certain storage reserve N_p is maintained.

Naturally, an attempt is made to see that the number of main file tracks not used in the current period is minimal, since this increases the number of procedures performed to manage the main file.



Use of Main File as a Function of Its Filling Rate. 1, Filling factor; 2, cylinders per day; 3, days.

Figure 5 shows the sequence of processing of user requests for retrieval of archive data. This technology provides rapid access to the data frequently accessed by users, requiring extra time to retrieve long-term storage data.


(5)
1 Перенос данных по запросам
2 Запросы пользователей
3 Основной файл 1-го уровня
4 Основной файл 2-го уровня
5 Запросы
6 Архив на МЛ

Processing of User Requests for Data Access. 1, Data transfer in response to requests; 2, user requests; 3, first level main file; 4, second level main file; 5, requests; 6, magnetic tape archive.

The computer center of the Institute of Cybernetics, Ukrainian Academy of Sciences, has developed an interface with a time-sharing system supporting dialog interaction of users with the ARKHIV software package. Additional capabilities for sanctioned access to user data are supported. The group processing functions of modules are expanded, and modules with the same names can be stored. The collection of statistics and analysis of the main file status are automated. We list below the command functions implemented in the ARKHIV software package.

1. Write, read, delete, change characteristics of a module in the main file.
2. Determine characteristics of a module or list of user modules.
3. Print a module.
4. Record library data set segments in the main file.
5. Replace main file module with library data set segments.
6. Select modules from the main file per characteristics.
7. Control message output.
8. Generate a date stamp.
9. Generate a list of library data set segment names.
10. Output a list of instructions and their modes of execution, authorized for the user.
11. Generate reference information.

12. Redefine an input instruction sequence: Sequential data set, main file module, central console.

Batch mode access to the system supports storage of large volumes of data in the archive, with the instructions controlling information processing in the input stream. A user operation protocol is produced as a library data set segment, or sent to the system output.

When operating with the ARKHIV software package by time sharing, input of instructions and output of messages and results are performed directly through the user's terminal.

The technology for maintaining the automated archive includes several stages of procedures:

— Creation of the main file;

— filling of the main file in on-line or batch mode;

— storage of the main file over a predetermined period;

— daily collection of statistics on the functioning of the archive, used to make decisions concerning transfer of modules from the operational archive level to long-term storage;

— transfer of information to the main file from magnetic tape on request;

— recovery of the main file.

The technology supports:

— Multilevel organization, allowing both operational access to data and requests for data in long-term storage on magnetic tape;

— storage of large quantities of data, data compression, protection of data access;

— documentation of data modifications, storage of information on modules;

— access to data from user application programs and application packages;

— effective data management: Storage, reproduction, recovery;

— generation of statistical data are various detail levels concerning operation of the archive, with generation of decisions based on these data concerning the level of filling of the archive, the possibility of transferring data from one archive level to another.

The use of the ARKHIV software package has liberated more than 60% of the disk memory and allowed effective data archive management.

UDC 658.5.001:658.514:681.3.01

Use of Data Collection and Preparation System Based on SM-1 Control Computer in Operations-Production Planning Subsystem of Enterprise Automated Management System

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian (manuscript received 08 Aug 86; after revision 25 Jul 87) No 3, May-Jun 1988, pp. 43-45

[Article by S. M. Grigorov]

[Text] The timeliness and quality with which management decisions are generated in the operational-production planning subsystem of an enterprise automated management system depend to a great extent on the speed and reliability with which data are entered into the computer. The technology used for this process at most enterprises assures slow management system reaction time. Usually, primary documents (accounting cards, receipts, invoices, etc.) are sent from subdivisions to the computer center, then the information from these primary sources is transferred to machine-readable media, or copies of the documents on punch tape generated as the primary documents are filled out on special equipment such as the RP-70 and RI-7501 are sent to the computer center. In either case, the data must be verified and corrected if necessary. This is followed by computer input and processing, after which the results must be delivered to the points where production is actually controlled.

The time cost of these procedures depends on the volume of information and the extent to which it is prepared for machine processing. In large enterprises, a daily cycle is typical, with documents sent to the computer center at the end of the first shift, verified and transferred to machine-readable media during the next shift, and processed in the third shift. The results are returned by the beginning of the first shift on the next day, after which they must be analyzed, serving as the basis for management decisions. This algorithm does not consider errors in input information which may not be found, or errors introduced at any stage in the process, leading either to incorrect decisions or to reprocessing. Obviously, neither situation satisfies production.

In practice a management decision, though not always well-founded, is made significantly earlier. The automated management system, particularly the operational-production planning subsystem, has little impact on the management loop and the material and labor resources expended do not yield the desired effect.

Thus, the pressing nature of the problem of decreasing automated management system reaction time while increasing information reliability and simultaneously decreasing labor consumption in the data collection process is obvious. It is virtually impossible to solve this problem without creating a task-oriented hardware system which would allow information to be input throughout the working day as it is generated directly at working locations, rigidly verifying the input process, performing simple processing and outputting results at the point of application in a mode as close as possible to the actual flow of the controlled process. The use of such a system could minimize the probability of errors, liberate data input operators at the computer center, reduce labor consumption of the entire process and allow data to be generated concerning the course of the production process at any time.

Many enterprises in our country (KamAZ Association, the Nizhnetagilsk Metallurgical Combine, etc.) are now using a data collection and preparation system based on an SM-1 control computer, which largely satisfies these requirements. The system includes a somewhat expanded SM-1 plus data transmission equipment to provide communications with information display panels directly in the production shop. Communications between the central computer and these displays are by dedicated telephone lines through the plant exchange. The maximum distance from the computer center is up to seven kilometers, with up to eight displays connected to a single telephone line.

The total number of network users is determined by the technical capabilities of the computer, the operating system used and the complexity of preliminary information processing. As a rule, the number of displays simultaneously controlled is not over 32 (at KamAZ).

The most commonly used displays are the RI-6401 and RI-6402. The capabilities of the latter model are somewhat expanded by inclusion of a "Konsul" input-output device. In addition to keyboard operations, both models allow data to be input from a standard punch card, with visual monitoring and editing off line. The "Stroka-190" printer used with the RI-6401 base model was not found to be successful and is therefore not provided or is replaced by a more reliable device of the same type manufactured for use from the "ELKA-55" Bulgarian calculator (manufactured by "Izhmash"). Communications over standard telephone lines are supported using APD-MPP data transmission equipment. The communications lines from the displays converge at the telephone exchange, and are combined into group channels which are connected to the computer through central exchange data concentrators.

It should be noted that this connection does not agree with the technical conditions for operation of the APD-MPP equipment, although with moderate traffic levels the equipment operates quite satisfactorily, with operator intervention required no more than two or three times per day for a transmission rate of 4800 Baud. When traffic is heavy, "receive" mode may be briefly blocked upon completion of "transmit" mode operations, resulting from the half-duplex operation of the APD-MPP equipment. The block signal is not sent to the display panel, creating the illusion that the panel has failed and causing operating personnel working with the devices to complain justifiably.

The SM-1 minicomputer has well developed software based on DOS ASPO and DOS RV for the creation of software-hardware systems capable of taking over a large portion of the routine functions of the supervisor section of a large shop or an entire production facility.

This type of system has been implemented at the "Izhmash" association motor vehicle plant. It is designed for timely accounting and monitoring of the course of production as a part of the plan-supervisor service.

The software, based on DOS ASPO, was developed for an information network consisting of 24 type RE-6401 displays. At present, four displays are in operation in two shops, supporting the main assembly conveyor. Work is now under way to connect five additional shops (10 displays). The system operates in basic production mode — 16 hours per day. In particularly heavily loaded sections, in order to assure complete reliability, displays are duplicated, which does not require additional communications lines.

Information collection is organized according to the model data input principle. The constant information for each shop, including the characteristics of a part or assembly unit, its path in the technological cycle, etc. is coded on punch cards. Files of 300-700 punch cards are created and periodically altered within the shops. The recording procedure consists of reading a punch card and indicating by keyboard input the volume of the batch of parts transmitted. This work is performed by the shop supervisor, and is usually not the most important responsibility on his list of duties. The result of this operation is entry of the information on machine-readable media and printing of the accompanying document at the point where the information was recorded. The time required to perform one operation, including locating the proper punch card, is not over 40 seconds, allowing the peak load at the middle and at the end of each shift to be managed. The intensity of operation of some of the information sources at this time is as high as 75 messages per hour, with a total information volume of 270 messages per shift.

The input is rigidly checked for syntax errors. Erroneous input is rejected, with an appropriate message and invitation to repeat the entry. Semantic checking is the responsibility of the human operating the display, who is provided with a broad arsenal of editing capabilities both during the shift and during any day of the current and following month. Naturally, this is done only with the permission of the planning-supervisory department.

The number of available standard inputs is quite great, allowing data to be input not only for timely monitoring of the course of production, but also, for example, data on standards and reserves of materials, their consumption or the organization table. There are seven models in this system, two of which are service models used to test the displays and their individual components without disconnecting them from the central computer.

Information sent over the communications channels is recorded on magnetic disk and duplicated on magnetic tape to allow rapid data recovery in case of failure of a disk drive or other problem during subsequent processing, even at the highest functional level of the automated management system. At the same time, documentation is printed out at the information display, i.e., at the point where information is generated — an invoice in two copies.

The operation of the entire system is monitored by a single operator, the computer center supervisor, along with his primary duties. In particular, on request by interested services he can output the information stored during a given portion of a day in the YeS computer, where the major preparation of the enterprise operating summary is performed. The time spent on this procedure is not over one-half hour. It is usually performed three times per day so that shop supervisors can be provided with reliable information before their shift begins. At the end of the day all accumulated data are output to the automated management system data base for use by other subsystems. Communications with higher level computers are supported through exchange of magnetic tape.

A supervisor system with a similar data collection section has been put in experimental use, managing a finished product storage area at a tool production facility. This system performs the following tasks:

— Creation and management of an internal data base;

— input of operational information;

— editing of the internal data base;

— generation of output documents with standard formats;

— generation of reports on user request in dialog mode, including incomplete request forms.

The software runs under DOS RV-2, the main advantage of which over DOS ASPO is the ability to swap active tasks, i.e., essentially an operating mode with practically unlimited virtual memory, quite important considering the limited RAM capacity of the SM-1 computer. This significantly expands the range of tasks performed directly in the minicomputer, and will allow easier increases in system capacity, particularly following the planned shift to a more modern type SM-1210 computer.

The data collection devices used are four type RI-6402 displays connected in a radial network, while the remote access devices used for the internal data base are two type DM-2000 displays.

This network structure increases the throughput capacity of the communications channels and significantly improves the viability of the entire system by minimizing the influence of APD-MPP switching time from transmit to receive and vice versa in half-duplex mode. The general contractor for development of software is "Uralsistem" Scientific-Production Association, whose specialists have a great deal of experience in developing supervisor systems of this class. Unfortunately, centralized manufacture of specialized hardware systems for remote collection and processing of operational production information, construction of middle-level supervisor systems such as the YeSTEL system with new generation peripheral equipment such as the SM-1604 and SM-1605, manufactured in Bulgaria, has still not been undertaken in our country. This situation forces us to adapt various hardware and software facilities for these purposes, which is not always possible at optimal cost. Doubtless, such systems should be based on the SM computers, allowing the creation of inexpensive, in many cases independent and quite flexible automated management subsystems, thus improving the effectiveness of automated management systems and, as a result, improving the effectiveness of production.

UDC 681.3.06

Accounting for the Use of Machine Resources in OS YeS

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian (manuscript received 15 Sep 86; after revision 17 Mar 87) No 3, May–Jun 1988, pp. 45–48

[Article by A. E. Ruzin and A. I. Khokhlov]

[Text] The extensive use of YeS computers as shared access machines makes the development of a methodology and software for accounting for machine resource utilization a pressing task. An accounting subsystem should perform three types of tasks[1, 2]:

— User accounting;

— reporting of the user load based on various characteristics;

— estimation of the usage effectiveness of an installation and determination of bottlenecks for future optimization.

The difficulty of the task results from the essentially multitasking ideology of the YeS computer operating system. Most accounting subsystems use as their information source the records generated by the system monitor program[3], the essential shortcomings of which[2, 4] generate additional difficulties.

This article describes the ideas forming the basis of a subsystem for accounting for certain machine resources developed at Donetsk State University, distinguishing this subsystem from others developed earlier.

System structure. All previously developed accounting subsystems (in any case, those known to us) have some pretentions to universality. However, almost always when such subsystems are introduced at a specific computer center, the software or data processing technology used at the computer center (or, still worse, both) must be changed. The reason is that the accounting subsystem, in addition to accounting itself, must perform the functions of generating initial information and printing out resulting documents. This makes the structure of the accounting information, as well as the composition and form of output documents, relatively rigid. This rigidity differs in various subsystems, but is always present.

This rigidity can be decreased by separating the aspects — the accounting subsystem performs only the accounting function, while the structure of the input and output information is determined by resources which are external to the accounting subsystem. Information on the use of machine resources over a given time interval can naturally be accumulated in one of the data bases used at the computer center. With this approach it is just as natural to use DBMS facilities to determine the composition and structure of the input and output information. Introduction of such a system requires minimum expenditures.

The subsystem we describe is constructed as follows. Two functionally separate parts are distinguished — we call one of these the input task, the other the output task. The input task extracts necessary information from system monitor program records, sorts them and enters the information in the data base[4, 5]. An operational summary is also formatted and printed, a table containing information on all tasks executed in a shift (something like a machine journal).

We utilize the INES DBMS, the most convenient for the Donetsk State University Computer Center. The tree of the data base includes embedded key files. The keys are the computer type, OS, user accounting information, year and month. The leaves contain data on the use of such resources as central processor time, disk access, punch cards read, lines printed, etc., plus an overall characteristic — commercial time.

This information is stored for some time (at least one month). When a job by a certain author is first executed, it is automatically recorded in the data base by the DBMS. In other words, a new branch on the data tree is created, and the values of the characteristics mentioned above are recorded on its leaves. If the author of the job is already recorded, the values of the leaves are modified.

Entry to the data base is performed using the INES DBMS input model system (any other DBMS could be used — hierarchical data organization is not necessary). In this stage, the structure of the accounting information is insignificant — it is simply used as a key[6]. The entry task can therefore be run at any computer center with any accounting information coding method already in use without modification. Furthermore, by removing the call to the INES analyzer (used to write to the data base), the input task can be used independently to generate the operational summary. Experience has shown that in some organizations this machine document is quite sufficient.

The second part of the subsystem, as we noted, consists of the output task, i.e., generation of output forms for various subsets of users, departments, subjects, etc., or for the system as a whole or the installation over a certain (usually predefined) accounting period. It is assumed[5] that, using the facilities of a well-developed DBMS (in this case INES, more precisely the query language and model output system), the required set of output forms can be easily obtained, based on the specifics of each computer center. Actually, the query system allows selection of data in the data base, interpreting key subfields in a wide variety of ways. The authors have developed a set of output forms considering the peculiarities of the operation of a computer center at a large higher educational institution in accordance with the structure of educational information used. Since the introducing of the subsystem, the information processing technology at the computer center has changed several times and new requirements have been set for output forms; in particular, requirements for new forms have been generated; each time these tasks were performed at minimal expense.

Commercial time. Penalties. The basic characteristic of the utilization of computer resources by tasks is the so-called commercial time. This characteristic is computed as a function of all the basic resources utilized by a task and is a more or less objective characteristic of the task, little dependent on the conditions under which it runs (i.e., on multitasking level, computer operator experience, etc.). However, the specific composition of resources accounted for, type of function and weight coefficients are not the best of all possible characteristics; these problems are worthy of individual and detailed analysis from the economic standpoint.

We note, however, that waiting time is not a characteristic of a task (since it includes waiting time for both its own and for outside events and, still worse, computer operator reaction waiting time), and is therefore not accounted.

In order to stimulate efficient utilization of computer resources by users, a system of penalties is used. Improper utilization of memory and class assignment is penalized (at a percentage of commercial time).

Furthermore, a user pays (in units of commercial time) for memory requested in the job control language operator, as well as the difference between the average memory allocated and actually used (if the difference is positive). Mean allocated memory is computed by the equation

$$M_k = \left(\sum_{i=1}^{N_k} a_{ik} t_{ik}\right) \Big/ \left(\sum_{i=1}^{N_k} t_{ik}\right),$$

while mean utilized memory is computed by the equation

$$L_k = \left(\sum_{i=1}^{N_k} b_{ik} t_{ik}\right) \Big/ \left(\sum_{i=1}^{N_k} t_{ik}\right).$$

where a_{ik} is memory allocated at point i in a task; b_{ik} is memory in use at point i in the task; t_{ik} is the time of execution of point i of the task; N_k is the number of points in a task.

Based on such objective characteristics of a task as the central processor time, number of input-output operations and volume of memory required, an agreement on task classes has been developed. It determines the mixture of initiators used with predetermined priorities. The agreement has been reduced to a list of all users, and in order to monitor its observation, the input task computes the class based on the actual utilization of resources, printed in the operational class summary, indicated in the JOB operator of the job control language, or the recommended class if it does not agree with the recorded class. Furthermore, in case of disagreement, a task is penalized in the form of a percentage of commercial time. In *MVT* mode this is apparently the only method which can force users to state their class properly. Users can learn the proper class to be assigned to their tasks by reading the operating summary.

Since the accounting system here described was introduced, the penalty system has yielded positive results. Thus, over the past year of functioning of the system, the memory utilization factor for j randomly selected tasks, calculated by the equation

$$X_j = \left(\sum_{k=1}^{j} \frac{M_k - L_k}{L_k}\right) / j,$$

where j=50 has decreased by half (from 0.11 to 0.056).

We have not estimated the effect of the class agreement and penalties for failure to observe it. We note, incidently, that introduction of the accounting system and all related steps has increased, for example, the multikasking coefficient of the YeS1022 computer by 14% (comparing characteristics for the first and 15th month of functioning of the system).

Individual access mode. There are at present two main machine resource access modes on the YeS computer — batch and dialog. At many computer centers with insufficient equipment capacity and, as a result, scheduling of computer operation, one or more users are allocated all resources (i.e., the entire installation) for monopoly use for scheduled time periods. The user (or group of users) can operate in batch or dialog mode during these time periods. We call this operating mode individual.

A user who has received permission for operation in individual mode is allocated a certain time in the schedule. During this time (which we shall call the individual access interval) the computer may actually stand idle (for example, while the user thinks over decisions) or may be virtually idle (while the user starts a task with the START instruction). Changes to the main planner allow virtual idle time to be accounted for, but only in the sense of estimating the effectiveness of utilization of the installation, since it is impossible to identify the user entering the START command.

On the other hand, if user tasks are performed in parallel, the total lengths of the tasks may exceed the length of an individual access interval, which cannot be permitted.

Finally, if the capacity of the installation is sufficient, in addition to the user operating in individual access mode, there may be another user, particularly a stream of batch jobs.

In order to solve all these difficulties, we suggest that the commercial time of the individual access interval be considered simply the duration of the interval in the astronomical time sense. Separate accounting for all resources becomes senseless if monopolistic operation on the entire installation is imitated in multikasking mode.

Accounting for a parallel stream of jobs is performed by the method described above.

With this approach a user clearly notes the beginning and end of his work (his interval) so that the system monitor program can record it and identification of the user is not difficult. This can be done, for example, by the use of a task with a special name. We shall refer to such a task as a password task. In the system we have introduced, the first character of the name of a password task is the character ⊗ (which was found to be most convenient from all points of view), while the remaining symbols contain accounting information. The content of the task is unimportant.

The individual access interval is defined as the time from the moment the first password task starts until the moment when the second (with the same name) starts or until the initial operating system loading procedure is executed. Therefore, if for some reason a user must reload the operating system (for example, due to a temporary power failure), operation must be started by inputting his password task.

The computer operator, monitoring the start of the password task with the name corresponding to the user number, is responsible for correct opening of the interval. The user himself is interested in correct closing of the interval.

Accounting in individual access mode is performed as follows. The input task recognizes among the system monitor program records the one which corresponds to the password task. The commercial time of the user whose accounting information (number) is contained in the password task is the entire length of the interval (in the sense of astronomical time). All nonpassword tasks with the same accounting information (number) running during the same interval do not change the "commercial time" field in the operational summary or the data base, but the total of their lengths determines the "task duration" field.

If in addition to the individual access mode there is also a parallel job stream (with other accounting numbers), it is considered independently. Herein lies the functional difference between individual access mode and assigned time mode in the DKP system[7], which frequently rejects tasks with accounting information different from the accounting information of the assigned time interval.

We note also that a program and procedure in job control language allowing opening and closing of the interval from the operators console have been developed for the convenience of users. The parameters of the procedure are the number and name of the programmer. The program generates a trivial password task and initiates its execution.

A sequence of system monitor program records can be similarly marked, thus sending the input task various types of useful information such as the name of the duty operator.

List of data sets and archive service. The accounting subsystem, in addition to the process outlines, maintains a centralized registry of permanent user data sets on direct access volumes. The magnetic data medium administrator usually executes the following tasks:

— Allocation and liberation of space on direct access volumes;

— deletion of contraband data sets, i.e., those not registered with the magnetic medium administrator;

— timely archiving of data sets and their recovery in case of damaged to the original.

These operations are automated using a special data set in a resident system volume. This set, called a list, consists of records, each of which describes a user permanent data set registered with the magnetic medium administrator. Each record contains:

— The registration number of the disk pack;

— the name of the data set;

— the name of its owner;

— the organization of the data set;

— the date of creation of the data set;

— the date of last closure of the data set with the INPUT parameter;

— the number of such closures;

— the date of the last closure of the data set with the parameters OUTPUT or UPDATE;

— the number of such closures.

Records in the list are sorted by disk pack registration number, data set name and owner name.

When a newly formed data set is registered with the magnetic medium administrator, the administrator forms a new record in the list, entering the first three fields. The remaining fields are automatically generated and maintained by the accounting subsystem input task using system monitor program records 14 and 15.

Simple list management programs have been developed. These programs allow printing of the list, addition of new records, as well as deletion or editing of old records. Examining the list, the magnetic medium administrator can make conclusions concerning the utilization intensity of various data sets.

The select program occupies a special position among the list management programs. It generates on disk a data set containing the names of data sets on the volume which have been changed since a certain date. This list of names is sent to the input of the archive storage system used at the computer center. Thus, only those data sets which have been changed since they were lasted backed up are archived. The use of this list, together with the INES archive system, significantly simplifies and speeds up copying of data sets into the archive, and also reduces the size of the archive on magnetic tape.

REFERENCES

1. Korotkevich, V. A. Maksimey, I. V., Shchers, A. L. "Organization of the Computing Process on ES Computers." Programmirovaniye, 1984, No. 1, pp. 87-92.

2. Treymanis, M. O. "One Class of Emergency Situations." ibid, 1983, No. 4, pp. 90-93.

3. Peledov, G. V., Raykov, L. D. Vvedeniye v OS YeS EVM (Introduction to the YeS Computer Operating System). Moscow, Statistika Press, 1977, 119 pp.

4. Ferrari, D. Otsenka proizvoditelnosti vychislitelnykh sistem (Estimating the Throughput of Computer Systems). Moscow, Mir Press, 1981, 576 pp.

5. Martin, J. Organizatsiya baz dannykh v vychislitelnykh sistemakh (Computer System Data Base Organization). Moscow, Mir Press, 1978, 662 pp.

6. Arlazarov, V. L., Yemelyanov, N. Ye. "General Description of the INES System." Problemy, MSNTI/MTsNTI, 1982, No. 1, pp. 46-59.

7. Panshin, B. N. "Problems of Development of the Program System 'Shared Use Computer Supervisor.'" USiM, 1979, No. 6, pp. 16-20.

UDC 681.3.068

The MP25 Macro Processor

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian (manuscript received 03 Apr 87) No 3, May-Jun 1988, pp. 49-52

[Article by A. S. Beketov and V. M. Smolkin]

[Text] **General information.** The MP25 macro processor (the MP25) is a member of the class of free universal macro processors designed to operate under the control of the ASPO operating system on SM computers[1]. The MGD macro generator, the language of which is an expansion of the basic MNEMOCODE programming language[2], has long been used under ASPO on SM computers. In contrast to the MGD macro generator, which may share some areas of application with the MP25, the latter has no limitations as to the nature of input and output texts. Texts may include, for example, computer programs in any programming language, books or a series of answers to a questionnaire. In this sense, MP25 allows maximum freedom of macro calls[3].

The macro processor has two inputs: Macro definition and input text. Each macro operation indicates, first of all, that which must be replaced and, secondly, what it must be replaced with. This is referred to as a text replacement macro. The input text is a sequence of letters in which replacements are to be made. If a certain set of macro definitions has been assigned, the first actions of the macro processor are inspection of the initial text and location of macro inputs.

Each such input is called a macro call or macro instruction. The macro processor, finding a macro call, determines the replacement macro text and replaces the call. The text replacing the macro call is said to be generated by the call. The output text is produced from the input text by replacing all macro calls which it contains.

Free entry of macros opens new paths for utilization of macro processors. Usually a set of macros is defined first, then programs are written considering these macros. The MP25 allows the opposite sequence of operations, i.e., utilization of macros to make changes in previously written programs. The use of macros to replace one sequence of letters with another throughout an entire program is referred to as context editing.

The need for context editing may arise under various conditions. The most common case is modification of programs received from elsewhere; even if written in a so-called standard language, it is usually necessary to make a few context replacements before a program can be used. For example, the rules for writing the FORMAT operator in certain dialects of FORTRAN allow a text line to be written as XYZ, while in other dialects it is necessary to write CHXYZ. Input-output operators are also frequent objects of local changes. The need for context editing arises also in the process of development of one's own programs. Typically, a number such as 20 might need to be replaced by the number 30 if encountered as the limits of an array or a loop, or when a new argument needs to be added to all calls for some program.

Macro processor operating modes. The MP25 operates in two modes: With and without confirmation.

In confirmation mode, MP25 requires that a macro call begin with the special character "#". We should note that the MGD macro generator, which requires that the name of a macro be located in the operation code field, can also be considered to operate with confirmation.

When operating without confirmation, i.e., without the warning marker, MP25 analyzes the input text line by line. Each macro has an associated pattern. The pattern is given as a line of arbitrary characters in which the formal parameter corresponds to "holes" marked with the parameter marker (the "*").

Each input line is compared in sequence with the pattern of each macro. An input line is recognized as a macro call if it and a pattern match precisely, character by character (with the exception of those places where the pattern indicates by a marker that the actual parameter should be located). Up to 22 parameters are permitted, which the macro replacement text (macro definition) can access as follows:

&0, &1, &2, ..., &9, &A, &B, &C, ..., &L

Let us look at an example, in which the input line

```
ALPHA=9BETTA+GAMMA)/DELTA
      IF (AB)3, 20, 7
```

is compared with the pattern

```
*=(*)/*
IF (*)*, *, *
```

In the first case, the actual parameters take on the following values:

&0=ALPHA, &1=BETTA+GAMMA, &2=DELTA

While in the second case:

&0=AB, &1=3, &2=20, &3=7

If there are less than 22 arguments, the remaining arguments receive null values.

The input line is checked by MP25 first in the mode with confirmation, and if this mode is not successful MP25 shifts to the mode without confirmation (i.e., performs search among the patterns).

In the mode with confirmation, macro definitions can be stored in a normal character file on disk, which avoids the use of special macro library editing programs. If neither mode processes an input line, it is copied to the output text with possible macro computation, i.e., execution of substitutions in the process of examination of the input text. Macro definitions are not copied to the output text. It is sometimes necessary to forbid macro substitution within certain contexts (such as rows of letters) which may be similar to macro calls or special macro language characters. Inclusion of letters in special brackets prevents computation of letters — they are simply copied to the output text without the brackets. This blocking mechanism allows any character to be inserted in the output stream.

Language elements. Each macro processor, including MP25, has its own programming language, with which instructions are given and which also serves to define and convert objects during generation of MP25.

The macro processor allows both local and global macro variables. The global variables are in turn subdivided into disk and ordinary. Global variables are required to transmit information from one macro call to another, while local variables are desirable for internal operations within individual macros. If recursion is possible, local variables are almost obligatory. Local variables have the following form:

&0, &1, &2, ..., &9, &A, &B, &C, ..., &L,

while ordinary global variables are

&M, &N, ..., &Z

The size of a local variable is limited to 26 characters, of a global variable — to 58 characters. In MP25 there is no special apparatus for assignment of macro variables; the variables, depending on context, may be literal, integer or boolean. For example, if we have the following variables: &1=12, &2=13, &N=TEXT, then the computation &1+&2+&N causes the output file to receive the line of characters 25+TEXT. Note the difference in the computation of the next line using the exclusion characters: &1+\&2\+&N. The result will be 12+&2+TEXT.

In case of a list of arguments of indefinite length, the macro call must have a mechanism to interpret the list. In MP25 this is achieved by means of computed macro variables, obtained from ordinary macro variables by attaching the character "&," for example: &&0, &&B, &&Z.

The MP25 macro processor allows the use of integer, decimal and octal numbers with no sign or with the "-" sign. The range of numbers is from -32768 to 32767.

Using the construction,

<macro variable> ' <macro variable or number>,

the user can work with indexed global disk variables. The macro variable or number following the apostrophe is the index of a variable (the index can be between 1 and 32767). The macro variable before the apostrophe may be a computed variable between &0 and &Z. Global disk variables have a maximum size of 26 characters and are not declared.

It is assumed that introduction of the following special variables has expanded the area of use of the MP25 macro processor: &% — the number of parameters in a macro call (determined from the macroinstruction); &U — a unique variable (incremented with each macro call); &F — the input file number; &Z — the number of the current input line; &* — the input line; &I — the index of a line found; &L — the remainder from a division, etc. The semantics of the special macro variables should be clear from their names.

Variables in MP25 may be of three types: Numerical, text and undefined. For the special macro variables, the type is fixed throughout the operation of the macro processor (with the exception of &*). However, ordinary and computed global and local macro variable types may change. In the initial state, the types of these macro variables are undefined. The type of a variable can be stated by adding the character "T" after the variable, for example: &0T, &TT, &BT.

An MP25 user may also find it useful to refer to the length of a variable. This is done by adding the character "L" to a variable, for example: &*L, &2L, &LL, &&1L. The operation of this apparatus needs no special explanation. Thus, if &3=SPRING, then &3L is equivalent to writing the number 6, since the length of the variable &3 is six characters.

To use the macro processor to generate lines of variable length, we must utilize the duplication of macro variables. The line

<macro variable> . <macro variable or number>

means duplicate the macro variable the indicated number of times.

For example, if &1=WINTER, &2=3, then the following are equivalent: &1.&2 and WINTERWINTERWINTER, &1.2 and WINTERWINTER.

It is impossible to work with lines of data without a mechanism to break up a line into its component letters. Referring to a macro variable with a colon and two numerical parameters, the user can extract a substring. The first parameter indicates the starting position of the substring, the second — its length. For example, if &1=FINAL, &2=2, &8=3, the following are equivalent: &1:3:1 and N, &1:&2:&8 and INA, &1:1:1 and F, &1:1:5 and FINAL.

To perform computations in MP25 language, we use the concepts of the operand and operation. The concept "operation" includes the following operations: Addition (the "+" character), subtraction (the "-" character), multiplication (the "*" character), division (the "/" character). As it reads text, MP25 always attaches the "operand–operation–operand" triplet to a single operand, performing the requested operation. For example, if &A=4, &B=12, when it encounters the text &A×7TEXT&B-&A+3, it is converted to the character string 28TEXT11.

We should note that interpretation of such a triplet always results in a decimal number. Naturally, the user can forbid an operation with the exclusion characters, for example: &B\+\&K. All arithmetic operations are performed left to right with no particular precedence, for example &2+3×6+5/7 will set the variable &1 equal to 5. The value and type of ordinary and computed global (disk and ordinary) as well as local macro variables can be changed by assignment operators: &A=TEXT+&A+3, &B=3+14×&C, etc. The macro variable to the left of the "=" is assigned the computed text to the right of the "=." The type of computed text will be numerical if as a result of the indicated convolution the text is reduced to a single number. No limitations are placed on the structure of the computed text, i.e., all macro variables (including special variables) may be contained, and the apparatus of convolution, duplication and substitution can be used along with their characteristics.

Another form of the assignment operator &A:=TEXT, using ":=" as the assignment symbol, is conditional and is processed only if the macro variable to which the text is assigned is of undefined type. This form of the operator is convenient for use to assign values to subsequent macroinstruction parameters.

A characteristic feature of many macro languages is full independence of the generation process, i.e., the user cannot change the values of macro variables from outside the language. This is inconvenient for debugging of macro definitions, which also must be debugged like an ordinary program. In MP25, the operator for communications with the user allows the user to interact more actively in the process of generation:

&[<text>]<macro variable>

The structure of the text is arbitrary and is printed at the operator's panel after it is computed. In response to this message the user inputs the text to be assigned to the variable.

Any line in a macro definition (except for a macro comment) can be marked with the macro flag, which can be referred to in conditional and unconditional transfer operators.

The user can change the sequence in which macro definition records are processed, using the operator &±; <macro flag> or &± <macro text>.

In the former case, MP25 computes the macro flag and processes the operator marked with the flag (top or bottom). In the latter case, the operator located the indicated number of operators above or below the unconditional jump operator is processed. We note that in this case the macro processor provides a facility for a multiposition transfer using the GOTO operator. This is quite useful, since multiple version processing is quite widespread for generation of substitution text to replace a macro call. The ability to compute a macro flag allows the subroutine apparatus to be used. When the address of a transfer is indicated outside a macro definition, the next operator following the unconditional transfer operator is processed, allowing the programmer to detect error situations.

By adding a relationship test to an unconditional transfer, the user can construct a conditional transfer of control. If the relationship evaluates as true, the conditional operator is processed as unconditional, if false — the next macro definition record is processed.

A macro definition may be a file on disk or the input file before the first macro instruction which accesses it. It is written as follows:

&(<beginning of macro definition>
<name of macro definition>
<body of macro>
&)<end of macro definition>

If the beginning of a macro definition is written as &(PA, the macro definition can be accessed following the rules for a pattern. Otherwise, the macro call should be accompanied by the special character "#".

It is convenient to use the concept of macroinstruction embedding level. If there is no macroinstruction, the level corresponds to zero. If a macroinstruction is encountered as a level n macroinstruction is interpreted, it is at level $n+1$. At level one, the macro processor can distinguish macroinstructions written in macro definitions as patterns or written with the special character "#". At subsequent levels, a macroinstruction can be used only with the "#" character. At level one, text is not computed, while at other levels all text is computed, even the names of macroinstructions.

Unfortunately, it is not within the framework of this article to provide a full picture of the capabilities of the MP25 language; therefore, we note in concluding our description of the elements of the language that the language includes operators for copying and deleting files, reassignment of logical numbers, boolean and numerical transformations, macro variable input and transformation, etc.

Debugging facilities. The debugging facilities allow the user to debug any macro definition in MP25 language, located either in a macro library or within the body of a program. These facilities are provided by the SOM macro definition debugger (a macro processor operating in debug mode) and were described in detail in[4]. Let us simply list some of the functional capabilities of the debugger: Read and write variables, record macro call breaks, record execution of macro definition fragments in single step and automatic modes, and if in trace mode, execute conditional breaks, etc.

Use of MP25. The MP25 macro processor has been in successful use for more than six years both by systems programmers and by users writing application programs within ASPO for the SM computer. The use of the macro processor has shown that MP25 is most effective as a means to assure portability of programs among computer models. Portability is assured by the fact that macros written in MP25 represent a new language for writing and debugging of program algorithms, requiring only reprogramming of the macro definitions for transition to a different model.

This method has been successfully implemented in planning of the #BMSP system programming macro library language[4], widely used to write system processing programs, and also to write the print format macro library (#BMFP) and the decimal arithmetic macro library (#BMDA). This same property of the macro processor was reflected in the creation of the AS31 structured assembler, facilitating further automation of programming labor and increasing programming productivity by 5–6 times in comparison to programming in low-level languages.

REFERENCES

1. Ayzenberg, A. B. "Development of Multifunctional ASPO Operating Systems." USiM, 1980, No. 5, pp. 57-59.

2. "Linguistic Programming Facilities For the SM-2 Computer Architecture." V. G. Vinokurov, V. M. Smolkin, V. S. Khayet and T. N. Yakusheva. Vsesoyuz. nauch.-tekhn konf. "Opyt razrabotki i vnedreniya tekhnicheskikh i programmnykh sredstv SM EVM i ASVT PS" (National Scientific and Technical Conference "Development and Introduction of SM computer and ASVT PS Hardware and Software Facilities." No. 2, 1986, pp. 26-28.

3. Braun, P. Makroprotsessory i mobilnost programmnogo obespecheniya (Macro Processors and Software Mobility). Moscow, Mir Press, 1977, 253 pp.

4. "Multifunctional Macro Assembler Level Software Debugging System (TEMP)." V. M. Smolkin, L. P. Volokh, V. P. Klimenko, et. al. USiM, 1986, No. 4, pp. 33-37.

UDC 681.3.06:658.012.011.56

Automated Testing of Modular Program Structures On Minicomputers

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian (manuscript received 03 Nov 86; after revision 12 Feb 87) No 3, May-Jun 1988, pp. 53-55

[Article by A. I. Sbitnev and A. M. Reznitskiy]

[Text] Minicomputers presently in use in automated management systems have large volumes of main memory and great speed. This has reduced the significance of the problem of optimizing programs from the standpoint of computer resource utilization, while increasing the importance of reducing the time required to develop and introduce automated management system special software. Broad utilization of high-level languages available for minicomputers (FORTRAN, PASCAL) and programming methods permitting improvements in programming technology have greatly increased the productivity of programmers' labor. Methods of structural, top-down and modular programming can be used to develop automated management system special software for minicomputers[1-3].

In this work, we shall analyze the problem of testing connections between modules in the planning of programs by modular programming methods, suggest a method for testing modular program structure and a hardware facility to support the method.

The process of program development by the modular programming method includes several stages. The first stage is decomposition of the program from the top down and construction of its modular hierarchical structure.

In the second stage, the programming languages are selected for the modules in the modular hierarchical structure, the modules are described in the languages selected, and connections between modules, both data and control, are planned. This stage is completed by independent testing of modules and elimination of algorithmic errors. Module testing methods have been described in many Soviet and foreign publication such as[4, 5].

The third stage involves composition (assembly) of the program from the debugged modules, its testing and debugging. One of the most important problems in this stage is organization of connections among modules, which may be written in different languages, based on the planned interfaces. The methods for solution of this problem were described in[6, 7].

Automated management system special software may consist of dozens of modules developed by a team of programmers. This leads unavoidably to errors in planning of data and control connections between modules. These errors are usually located in the stage of assembly and combined debugging of the program, since independent testing of modules cannot reveal errors of this type. The discovery of connection errors at this late stage in development sometimes has extremely unfortunate results — right up to complete replanning of the program. It would therefore be preferable to test the modular hierarchical structure of a program based on the specifications of the planned modules before independent module testing.

The method we suggest for testing the modular hierarchical structure of a program consists of testing the truth of a series of statements concerning the correctness of connections among modules.

The implementation ratio, written R_d, refers to the interaction among modules generated by functional decomposition. The expression $m_i R_d m_j$ then means that module m_j is a result of decomposition of module m_i. Thus, the modular hierarchical structure (MIS)=$<m_0, M>$, $m_0 \notin M$, where m_0 is the main module of a program, M is a set of modules for which the following statements are correct:

$$\neg \exists m_i \in M \mid m_0 R_d m_i.$$
$$\forall m_j \in M \Rightarrow \exists m_i \in \text{МИС} \mid m_j R_d m_i.$$

Let us study the relationships among modules, representing the control and data connections.

Transfer of control is performed when a module is called for execution. Upon completion of execution of a module, it must return control to the calling module. Let us represent the call and return relationships as R_c and R_r. Data connections are implemented by transferring data by means of parameters during calls and returns, and also through nonlocal facilities[8]. Let us represent the relationship involved in transferring data through parameters during a call and a return as R_c^p and R_r^p. The relationship generated by a data connection using a nonlocal facility we shall call a direct access relationship, represented as R_a.

For large programs consisting of many modules, the following control connection planning errors are typical:

— The structure includes modules which are never called (we shall call them isolated modules). The presence of isolated modules contradicts the statement following from the definition of the modular hierarchical structure:

$$\forall m_j \in M \Rightarrow \exists m_i \in \text{МИС} \mid m_i R_c m_j; \qquad (1)$$

— there are calls to modules which are not defined in the structure (we shall call them dangling references). The presence of dangling references contradicts the statement, flowing from the definition of the modular hierarchical structure:

$$\forall m_i \in \text{МИС} \mid m_i R_c m_j \Rightarrow m_j \in M; \qquad (2)$$

— verticality of control is violated in a structure, contradicting the statement

$$\forall m_i,\ m_j \in \text{МИС} \mid m_i R_c^t m_j \Rightarrow \neg\, m_j R_c^t m_i, \qquad (3)$$

where R_c^t is the transitive call relationship, defined by the following statement:

$$\forall m_i, m_k, m_l \in \text{МИС} \mid (m_i R_c^t m_k) \wedge (m_k R_c m_l) \overset{\text{df}}{\Rightarrow} m_i R_c^t m_l.$$

Data connection errors may arise during planning of exchange of data through parameters or through a nonlocal facility. At the modular hierarchical structure testing level, the correctness of the data connection between two modules utilizing data transfer through parameters is tested by comparing the number, type and sequence of parameters sent and received:

$$\forall m_i, m_j \in \text{МИС} \mid m_i R_c^p m_j \Rightarrow P_{i(c)} = P_j^{\text{in}}, \qquad (4)$$

$$\forall m_i, m_j \in \text{МИС} \mid m_j R_r^p m_i \Rightarrow P_j^{\text{out}} = P_{i(r)}, \qquad (5)$$

where $P_{i(c)}$ are the data sent by module m_i through parameters in the call to module m_j;

P_j^{in} are the input data of module m_j received through the parameter;

P_j^{out} are the output data of module m_j, transferred through parameters;

$P_{i(r)}$ are the data received by module m_i through parameters as control is returned from module m_j.

The correctness of planning of data connections using a nonlocal facility is tested by comparing the names and types of data declared in the modules as global and external, respectively. To improve the clarity of the texts of modules and facilitate testing and debugging, the program developer must meet certain planning requirements[2]:

— A module must have one entry and one exit, the entry point identifier must be the same as the name of the module;

— variables specifying results must follow arguments;

— global data must be declared only in the main program module.

This last requirement allows us to define the condition of direct access to global data by the following statement:

$$\forall m_i \in M \mid (G_i^{\text{in}} \cup G_i^{\text{out}} \neq \varnothing) \Rightarrow (m_i R_a m_0) \wedge (G_0 \neq \varnothing) \wedge \\ \wedge ((G_i^{\text{in}} \cup G_i^{\text{out}}) \in G_0), \qquad (6)$$

where G_0 are global data declared in the main module; $G_i^{\text{in/out}}$ are external data used in module m_i as input–output data.

The Kiev Institute of Automation has developed a hardware facility to support this method of testing the modular hierarchical structure of a program — ISP ANTES (structure analysis and testing). The initial information used to automate program modular hierarchical structure testing includes the specifications of the modules forming the modular hierarchical structure. The specifications of a module are developed from the text of the program and include the following information: Name of module, input–output parameter descriptors, global data used, as well as descriptors for connections with external modules. ISP ANTES consists of four programs written in PASCAL for DOS ASPO SM-2.

The BAZA program allows creation of a module specification data base in either dialog or batch mode. Then, based on the information contained in the specification data base, control connections are tested by the UPR program. This program constructs a graph of the structure of the program with respect to calls and checks for dangling references and isolated modules based on (1) and (2).

If there are no dangling references or isolated modules, the UPR program checks the principle of verticality of control using (3). The UPR program provides the user with the following information:

— Composition of the module specifications data base in the form of a list of names;

— a graph of the calls in the program in the form of a list of lines on the graph;

— names of modules containing dangling references;

— identifiers of dangling references and names of isolated modules;

— loops in the control graph in the form of chains of module names which violate the principle of control verticality.

If the UPR program finds control connection errors, the results of its operation must be analyzed and the corresponding corrections made to the specifications data base in dialog mode using the specification text editor program REDAT. Control connections are tested and corrected again until all connection errors in the modular hierarchical structure specifications are eliminated. After this, data connections can be tested by the program DAN, which constructs graphs of data transfers using parameters with calls and returns, as well as a graph of accesses to main module global data, and tests data connections for each pair of modules, represented as a line on the graph. Testing is performed by means of statements (4)-(6).

The DAN program provides the user with the following information:

— Graphs of the corresponding relationship, in the form of lists of lines on the graphs;

— pairs of modules in which data connection errors were detected;

— pairs of modules written in different languages which have data connections.

The results of operation of the DAN program can also be used to correct module specifications in the data base, which is followed by further testing until all data connection errors in the module specifications are eliminated.

The specifications of modules making up the modular hierarchical structure of a program, with corrected control and data connections, can be printed by the REDAT program and used in subsequent automated management system special software development, as well as software maintenance.

REFERENCES

1. Mayers, G. Nadezhnost programmnogo obespecheniya (Software Reliability). Moscow, Mir Press, 1980, 360 pp.

2. Khyuz, J., Michtom, J. Strukturnyy podkhod k programmirovaniyu (The Structural Approach to Programming). Moscow, Mir Press, 1980, 278 pp.

3. Glass, R. Rukovodstvo po nadezhnomu programmirovaniyu (Handbook of Reliable Programming). Moscow, Finansy i statistika Press, 1982, 256 pp.

4. Mayers, G. Iskusstvo testirovaniya programm (The Art of Program Testing). Moscow, Finansy i statistika Press, 1982, 176 pp.

5. Lipayev, V. V. Testirovaniye programm (Testing of Programs). Moscow, Radio i svyaz Press, 1986, 296 pp.

6. Lavrishcheva, Ye. M., Grishchenko, V. N. Svyaz raznoyazykovykh moduley OS YeS (Connections Between Modules Written In Different Languages Under OS YeS). Moscow, Finansy i statistika Press, 1982, 127 pp.

7. Grishchenko, V. N. "Software Facility Programming Problems." USiM, 1987, No. 1, pp. 68-71.

8. Pratt, T. Yazyki programmirovaniya: razrabotka i realizatsiya (Programming Languages: Development and Implementation). Moscow, Mir Press, 1979, 574 pp.

UDC 681.3.06:51

Software System For Diagnosing Microprocessor System Program Modules

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian (manuscript received 04 Nov 86; after revision 16 Apr 87) No 3, May-Jun 1988, pp 55-58

[Article by Yu. M. Korostil and V. A. Gulyayev]

[Text] **Introduction.** Process control tasks are frequently performed by computer systems including microprocessor-based computers[1]. In this case, the size of the software used with an individual computer usually does not exceed 10^4 machine-language instructions. Studies undertaken in[2] indicate that simple technological facilities should be used for development and maintenance of this software.

In this article we shall study a system of programs which represent an automated diagnostic system for software and program systems for the SM-1800 computer. The automated diagnostic system allows diagnosis of certain types of logical and functional errors and the logic of interrupt processing, and is intended for use with software written in machine-oriented language.

In comparison to known testing methods[2, 3], the automated diagnostic system uses methods which consist of a synthesis of symbolic, deterministic, top-down and other testing methods, modified in various ways. According to[4], the software is interpreted as a program product; therefore, the automated diagnostic system is a tool for testing of this product. The automated diagnostic system can locate a single error in a single pass.

The major task performed within the automated diagnostic system is maximum automation of the process by which a user implements procedures for test example data input and diagnosis of software modules and program sets.

Logical error diagnosis. By logical errors, we refer to errors consisting of incorrect utilization of the logical capabilities of the programming language used. Errors of this type can be recognized only if they lead to unsanctioned transfers of control outside the limits of a software module, storage of data in improper areas in memory or looping of software modules.

The following limitations are established to support diagnosis of logical errors:

— On distribution of modules in memory, when free memory areas must be left between modules, and programs and data must be placed in separate memory areas;

— on the method of implementing standard programming procedures, such as loops, data references, performance of logic functions, rules for stack use and transfer of control to other modules.

Limitations on the methods of implementing loops consist of the use of a limited set of loop predicates. According to[5], there are three types of loops (counter, address and associative), each of which, in addition to other instructions, contains an assigned sequence of instructions characteristic for any loop of the given type. These obligatory instruction sequences are called the loop predicates.

Limitations on the method of implementing data transfers consist of the requirement to perform transfers in loops containing only the transfer function. This is done using transfer predicates, the essence of which is similar to that of loop predicates.

Logic functions implementing the algorithm of the module must process one element of the input data set in one loop of algorithm operation. It is not recommended that the stack be used to transfer large volumes of data, but the stack address should be restored within each module, transfer of control to another module should be performed by control transfer instructions, using the stack to store the return address.

Unsanctioned control transfer is diagnosed in two stages. Before the software set is loaded, the automated diagnostic system fills the operating area of memory with instructions not used in the software set which transfer control to the interrupt area. After the software set is loaded, the automated diagnostic system, examining main memory, composes a table of software module and data addresses, replacing the starting address of called modules within each module by the operating addresses of the diagnostic module. After this, the automated diagnostic system starts the software set. Each module is started through the diagnostic module, which replaces the return instruction from the module with an instruction to transfer control to the module through the interrupt area. Therefore, in case of an unsanctioned transfer of control from one module to another, the diagnostic system can recognize a return from a module which it did not start.

In the second stage, the software module located is processed step by step, and the automated diagnostic system checks each address instruction for the permissibility of the address generated using the memory distribution table. Thus, the location of errors leading to an unsanctioned control transfer is performed to the instruction where the nonpermissible address is generated. These facilities are also used to diagnose cases where data are written into nonpermissible memory locations. The limitations on methods of data transfer allow the automated diagnostic system to detect an error involving storage of data in nonpermissible memory areas.

As each software module is started, a timer is also started and the time of operation of the module is recorded in the diagnostic module. The check time of a module is determined automatically by the automated diagnostic system using the number of instructions in the module and the quantity of input data. This method of determining a check time is capable of detecting times which exceed the normal time by an order of magnitude. With a less favorable relationship among the lengths of algorithms processing various portions of the input data, perhaps 1/10, and a ratio of input data processed by various software module parts of 1/100, for a module of 500 instructions length the check time might be 0.25 s instead of 0.025 s. The automated diagnostic system generates a table of the functioning of the software set, which is used in subsequent work.

Functional error diagnosis. Functional errors diagnosed by the automated diagnostic system cause the values of output quantities or their structure to fail to meet the requirements of the output specifications. Functional error diagnosis is performed by tracking the agreement of successively generated output structure elements with the elements of the output specification description, refined by the test example used. The process of tracking agreement in the automated diagnostic system is implemented by multitasking operation, with the software set being tested running simultaneously with a portion of the automated diagnostic system software performing step by step verification of the output specification.

Diagnosis and location of functional errors are performed in stages. If in the initial stage by the moment a fragment of the output specification is analyzed it is found that the software module has generated several elements which do not meet the requirements, the multiprogramming interval is reduced and the software set is restarted. This process is repeated until just one fragment fails to meet the specifications. At this point the automated diagnostic system starts to process the fragment of the software set located between the start point and the instruction in the software module after which the transfer was made to the automated diagnostic system module. This fragment is processed by sequential transition from the input specification to the output specification.

Sequential transition from the input specification to the output specification includes several stages. In the first stage, based on the current value of the test example, the software set is executed in single-step mode as the automated diagnostic system constructs a table of transfer addresses, which is a graph of the present implementation of the software set fragment. In this stage a table of input parameter transforms is generated, containing the altered value of each processed parameter along with the address of the processing instruction. If a looping fragment is located in this section of the module, two successive values are taken as the beginning and end of the loop. The second operand used to transform the parameter is written in the next line of the table of transformations executed.

The next stage of automated diagnostic system operation is based on the use of various algorithms to analyze the characteristics of the specification elements. Three types of specification element characteristics are distinguished in this approach: Numerical value, logical value and coordinate.

If a specification element is a number which in the test example at hand should have a certain value, then this value is a characteristic of the first type. If the specification element is a logical variable, the logical value is represented by the set of one bits in the byte or the number of shift positions between the codes of the required and actual byte (if the number of ones is the same). If the specification element should be located at an assigned position in the specification but operation of the program places it in another location, this means that it should be located at a certain distance in the output buffer from the previous element or at another distance from the start of the output buffer. This distance is a characteristic of the third type. Since the automated diagnostic system diagnoses only individual errors, only one of these characteristics will be analyzed.

In this stage, diagnostic operation involves processing of the reverse transform of the output parameter with the required characteristic.

Reverse parameter processing is based on algorithms which check for the existence of errors of a certain type in program fragments. In this case, the following types of errors are found:

— Functional transform instruction omitted;

— functional or logical transform instruction improperly used;

— erroneous parameter selected;

— second operand improperly selected[6].

The transform is performed as follows. Beginning at the bottom line of the table of transforms executed, the next parameter value is selected. The program fragment performing this transform is selected and the value of the characteristic of the parameter required is selected. This characteristic is known for the last step of the transform since the automated diagnostic system utilizes the output specification of the test example. Assuming that the error was a single error, it should be eliminated by correcting, adding or deleting one functional instruction.

The algorithm used is based on running through correction versions corresponding to possible types of errors. For example, it is initially tested whether a functional or logical transformation instruction was correctly used. This is done by replacing it with other instructions of the same group of transformation instructions and checking the current value of the parameter, taken from the next line in the table of transforms. If the processing generates the required specification element, it is considered that the error has been located; otherwise, the algorithm starts analyzing the next possible error.

If no version causes the parameter characteristics which are required to be generated, one step of reverse transformation of the elements of the output specification is performed, corresponding to the transformation of the program fragment in question. The transformed element value is stored in the table of required transforms and the next fragment of the program being tested is taken from the table of transforms executed. During this stage of diagnosis, a table of required transforms is generated, which is subsequently used to diagnose functional errors caused by more complex factors.

In contrast to traditional tracing facilities, the automated diagnostic system itself determines the software fragment to be displayed and can show the user current values of a parameter, values of parameters from the table of required values, and increase the level of automation of the process of diagnosing functional errors by adding more complex diagnostic algorithms. Experimental studies of the algorithms described above have confirmed their effectiveness.

Diagnosis of interrupt processing logic errors. Errors in interrupt processing logic lead to various temporary disagreements between presumed and actual software set fragment processing intervals or the intervals of individual modules, as well as incorrect sanctioning of software set interrupts.

A timer is used to model the necessary sequence or set of interrupts, the operating mode of which is set by a table of external actions affecting the software set. This table is filled in by the user considering the presumed algorithm of external interrupts. It indicates the interrupt level and time interval between an interrupt and the previous interrupt. As the software set is processed, the automated diagnostic system determines the flag indicating sanctioning of an interrupt of a given level in the interrupt table; if an instruction to allow or unmask interrupts is encountered, or if interrupts are inhibited or masked, the automated diagnostic system resets the interrupt sanction flag.

Each interrupt generated by the timer in accordance with the table of external actions is analyzed by the automated diagnostic system. If the interrupt corresponds to the sanctioned interrupt flag in the interrupt table, control is transferred to the interrupt handler. The reaction time of the automated diagnostic system to an interrupt is recorded and considered as the external action table is generated. If the interrupt sanction flag does not permit the interrupt which has occurred, it is not processed. In this case the address of the instruction in the module where the interrupt occurred is recorded, and after the module is processed the user receives a message indicating unprocessed interrupts, the coordinates of the module corresponding to the moment when such an interrupt occurred, and the address of the instruction which had modified the interrupt sanctioning flag. If desired by the user, when errors occur the processing of the module can be halted and diagnostic information generated.

Since the time relationships between the interval processing of various software modules and their fragments are most frequently regulated by the moment of arrival of external actions or the reaction time of the software module to input data, these automated diagnostic system facilities allow logical errors in interrupt processing to be tested with some approximation of the actual process.

Automated diagnostic system composition and functioning. The automated diagnostic system consists of four functionally oriented modules: Diagnosis, interrupt processing, functional error diagnosis and the control module.

The main purpose of the diagnostic module and the functional error diagnosis module was described above rather completely. The interrupt processing module diagnoses interrupt errors and consists of programs for operations with the timer, for processing subroutine calls, programs to control the interrupt sanctioning flag and analyze timer signals imitating external actions. The control module implements the dialog between the automated diagnostic system and the user, allowing passing of all necessary information concerning the software to be diagnosed to the automated diagnostic system and starting of the automated diagnostic system for diagnosis of logic, functional or interrupt processing errors.

When using the automated diagnostic system it is preferable first to diagnose and correct logic errors, then functional errors and finally — interrupt processing errors. The automated diagnostic system is designed for use under SRM? or OS-1800 on SM-1800 computers, occupies 12K of memory and can debug software sets of 20-30K.

REFERENCES

1. Mikprotsessornaya tekhnika (Microprocessor Equipment). Moscow, Voyenizdat Press, 1986, 111 pp.

2. Lipayev, V. V. Testirovaniye programm (Testing of Programs). Moscow, Radio i svyaz Press, 1986, 295 pp.

3. Mayers, G. Iskusstvo testirovaniya programm (The Art of Testing Programs). Moscow, Finansy i statistika Press, 1982, 176 pp.

4. Kulakov, A. F. Otsenka kachestva program EVM (Computer Program Quality Estimation). Kiev, Tekhnika Press, 1984, 168 pp.

5. Gulyayev, V. A., Korostil, Yu. M. "Software Facilities For Diagnosis of One Class of Errors In Software Modules On the SM-1800 Instrumental Computer." Gibridnye vychislitelnye mashiny i kompleksy (Hybrid Computer Machines and Systems). Kiev, Nauk. dumka Press, 1987, pp 12-23.

6. Teyer, T., Lipov, M., Nelson, E. Nadezhnost programmnogo obespecheniya (Software Reliability). Moscow, Mir Press, 1981, 324 pp.

UDC 025.43:681.3

ASAT-2 Network Terminology Analysis System

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian (manuscript received 03 Mar 87) No 3, May-Jun 1988, pp. 59-64

[Article by E. F. Skorokhodko, D. F. Podpolnyy]

[Text] **Purpose of the system.** The languages used with various types of automated systems (information systems, expert systems, etc.) include, particularly, special dictionaries, for example information retrieval thesauri. Creation of dictionaries for automated planning systems involves a number of difficulties related to the selection of a lexicon, its classification, etc. These difficulties are aggravated by inconsistent terminology, the disagreement and even contradictions found in the interpretation of terms in a single terminological system, formal logical errors (vicious circles in definitions, etc.). Elevation of the level of linguistic support requires preliminary analysis of the scientific and technical terminology to produce an objective evaluation of its properties and, where necessary, correct the list of terms and/or their definitions.

This analysis presumes the solution of a broad class of problems, which can be arbitrarily divided into four groups.

Isolation of terms which are in a certain relationship to an individual term. Problems in this group include: Isolation of terms which are directly contained in the definition of a term, i.e., its direct semantic components; isolation of terms through which a given term is defined directly or indirectly, i.e., all of its semantic components; isolation of terms, the definitions of which directly contain the term in question, i.e., its direct semantic derivates; isolation of terms defined through the term in question directly or indirectly, i.e., all of its semantic derivates; construction of chains passing through the term in question, in which each subsequent term is a direct semantic derivate of the previous term, i.e., chains of derivation (chains of minimal and maximal length, chains which begin with the term in question and chains which end with the term in question are particularly distinguished); construction of derivation chains relating to particular terms.

Isolation of terms in a given relationship to the entire terminological system. Tasks of this group include: Isolation of the most general terms, not defined in the terminological system in question, i.e., the initial terms; isolation of terms not included in the definitions of other terms, i.e., terminal terms; isolation of groups of terms and individual terms which are semantically unrelated to the basic terminology, isolated groups of terms and isolated terms; construction of all derivation chains relating the initial and terminal terms, including isolation of chains of maximum and minimum length; isolation of cyclical derivation chains (beginning and ending in the same term), i.e., vicious circles in the definitions of terms.

Determination of quantitative parameters characterizing individual elements of the terminological system. Problems in this group include: Determination of the number of direct semantic components of a given term; determination of the number of all semantic components of the given term; determination of the number of direct semantic derivates of the given term; determination of the number of all semantic derivates of the given term; determination of the length of the maximum derivation chain connecting the initial and the given term, i.e., the

D-order of the term; determination of the length of the shortest derivation chain connecting the initial terms and the given term, i.e., the K-order; determination of the number of all derivation chains beginning (or ending) with the given term.

Determination of quantitative parameters representing the terminological system as a whole. Problems of this group include: Determination of the number and specific gravity of initial terms; determination of the number and specific gravity of terminal terms; determination of the relationship of the total number of terms in the terminological lexicon to the number of initial terms, i.e., the derivation activity; determination of the maximum and mean number of direct semantic components, semantic components, direct semantic derivates and semantic derivates; determination of the maximum and mean D- and K-orders; computation of the connectedness factors of the lexicon[1]; determination of the number of all derivation chains connecting the initial and terminal terms.

In addition to these tasks, it must be possible to rank elements of the lexicon according to certain characteristics (listed in group three) and to edit the lexicon (change the number of terms, eliminate vicious circles, etc.).

In actual lexicographic sources (terminological dictionaries, collections of recommended terms, terminological standards, etc.), the information required is not explicitly stated, although it is present in implicit form. Its manual isolation for a dictionary of any significant size is an exceptionally difficult and labor-intensive task. The complexity of terminological analysis results from the fact that a terminological lexicon, like a lexicon in general, is not an amorphous conglomerate of words and word combinations, but rather a systemically structured formation in which each element, i.e., each term, is related to the remaining elements of the lexicon by a multitude of connections — direct and indirect. The number of connections and their length, or depth (the number of terms they encounter) are so great that they are practically impossible to trace without the use of automation equipment. This results in hidden errors which significantly influence the quality of a terminology, such as incomplete sets of terms, incomplete or inprecise definitions, vicious circles in definitions, etc. Consequently, a set of tools is needed to automate processes of analysis and correction of terminologies.

The Institute of Cybernetics, Ukrainian Academy of Sciences has created an automated system for analysis of terminological lexicons called the ASAT-2, designed to perform the tasks outlined above. The system supports analysis and editing of terminologies. It allows computation of more than 30 quantitative parameters characterizing the individual terms and the entire terminology, construction of a number of classifications and sequences of terms, locations of terms in certain semantic relationships with a given term, verification of the presence of semantic connections between given terms, etc. The terminology is tested for integrity, circles in definitions and certain other errors are located (and eliminated in interactive mode).

The system is intended for use by terminologists, automated system linguistic support planners, technicians planning banks of terms, etc.

In contrast to the experimental terminology analysis system ASAT created in 1979[2], the ASAT-2 system does not have rigid limitations on the size of the dictionary, number of direct semantic components and direct semantic derivates, the performance characteristics are significantly improved, the range of tasks performed is significantly expanded (particularly in the first and third groups); interactive-mode operations are allowed.

The network model of a terminological lexicon. The ideal model of any systematically structured formation (including a terminological dictionary) is a graph. A graph, the points of which represent the terms (more precisely, their meanings), while the lines illustrate the relationships among them, is called a lexical semantic network (a mutually unambiguous correspondence is observed between terms and points).

The traditional method of representing a lexicon, including a terminological one, is an interpreting dictionary, in which each term corresponds to certain other terms from its definition.

These defining terms are interpreted by us as direct semantic components with respect to the term defined, which in turn is interpreted as a direct semantic derivate with respect to the defining terms. In this case the lines of the graph reflecting the terminological lexicon are arcs directed from the points corresponding to the defining terms to the points corresponding to the defined terms (from direct semantic components to direct semantic derivate).

In this interpretation, the semantic network is an oriented graph. As a result, practically all of the relationships and parameters of the lexicon listed above have graph–theory analogues. For example, the semantic derivates of a certain term are nothing more than points which can be reached from the point represented by the term; the D–order of a term is the length of the longest path leading from a root point to the point of the term, etc.

Consequently, network modeling is an approach to the study of the properties of a lexicon as the properties of a mathematical formation — a graph, i.e., the terminological analysis tasks outlined above are reduced to graph analysis tasks. This permits broad application of graph theory methods.

Graph–theory methods for analysis of a terminological lexicon. Analysis tasks are performed in three stages. In the first stage, the tasks of the second group are performed. These tasks can be interpreted as collection and organization of information on the structure of the semantic network and isolation of substructures. The most common graph theory method for collection of this type of information is specially organized trip around the points and lines of the graph. The information thus generated is formulated as a set of numbers of the points.

Numeration F of the points of graph G is called an injective mapping of the set of points $V(G)$ of the graph in set N of the natural numbers: $F: V(G) \mapsto N$.

This information is generated using the so–called N numeration, based on a search to depth[3], one of the results of which is isolation of all loops in the graph. Loops in a semantic network correspond to vicious circles in the terminology, indicating, in general, incorrect definitions of terms. It is therefore desirable to eliminate such circles when they are found.

Elimination of circles in the early stages of operation of the system is useful not only from the contextual aspect, but also from the purely technical aspect. This is because, first of all, most graph–theory methods are most effective on acyclic graphs and, secondly, the elimination of circles facilitates the determination of initial and terminal terms. It is impossible to describe a terminological system correctly without locating terminal and particularly initial terms. In an acyclic graph there are always root and terminal or sheet points[4]. The former correspond to initial terms, the latter to terminal terms.

Two methods of eliminating circles are possible: In interactive mode, when the user himself edits definitions, i.e., the connections between terms; and in automatic mode, when one of the connections forming the circle is broken. We note that the automatic method of eliminating circles is not always correct, since the connections forming a circle are not equivalent. Its use must therefore be limited to the simplest cases.

After circles are eliminated, the semantic network is an acyclic graph. The *N*-numeration of the acyclic graph represents topologic sorting of the points[3], i.e., ordering of the points of the graph such that the contiguity matrix is an upper triangular matrix.

Topologic sorting has the following property: Let $T(v)$ be the number of point v after topologic sorting of graph G. Then if point v_2 can be reached from point v_1, then $T(v_1)<T(v_2)$, $(v, v_1,v_2 \in G)$. From this it follows that: If $v_1, v_2, \ldots, v_m$ is a path in graph G, then for $i=1, 2, \ldots, m-1$, it is true that $T(v_i)<T(v_{i+1})$.

We can conclude from this property that topologic sorting of the points on a graph yields the order of inclusion of the points in a path on the graph. Consequently, topologic sorting of points in a semantic network yields the sequence of inclusion of terms in a derivation chain. This result of topologic sorting simplifies the performance of the remaining tasks of analysis.

To perform tasks in groups one, three and four, information is needed on the connections between the individual elements of the lexicon. Therefore, in the second stage of analysis information is collected and organized on the system of connections in the lexicon. Information obtained as a result of the second stage of operation is formulated as special matrices.

In addition to the contiguity matrices, three more matrices are related to oriented graphs.

The reachability matrix D, recording the fact that one point can be reached from another (element $d_{ij}=1$ if there is a path from the point numbered i to the point numbered j, $d_{ij}=0$ otherwise).

The distance matrix S, recording the distance (length of shortest path) between points (element $s_{ij}=k$ if point number j can be reached from point number i, while the length of the shortest path between these points is equal to k, while $s_{ij}=\infty$ otherwise). Matrix S is sometimes called the shortest path matrix, since, using it, one can rapidly find the shortest path length between points.

The longest path matrix P, fixing the length (length of critical or maximum path) between points (element $p_{ij}=d$ if point number j can be reached from point number i, while the length of the maximum path between these points is d, $p_{ij}=\infty$ otherwise). Matrix P is sometimes called the critical path matrix since, using it, one can rapidly find the critical path length between points.

The matrices D, S and P are interconnected, i.e., $d_{ij}=0 <=> s_{ij}<\infty <=> p_{ij}<\infty$.

The reachability matrix is determinant, since most methods of its construction allow construction of the maximum length and distance matrices. The most common method of constructing these matrices is the method of Floyd[5].

Since the first stage of operation of the system involves topologic sorting of the points of a semantic network, this should be considered in developing a procedure for constructing the reachability matrix D from the contiguity matrix A by the method of Floyd:

```
FL: PROC (A, D, N);
    DCL (A(N, N), D(N, N)) BIN(1);
    D=A;
    DO K=2 TO N-1;
    DO I=1 TO K-1;
    IF D(I, K)=1 THEN
   DO J=K+1 TO N;
    IF D(K, J)=1 THEN D(I, J)=1;
    END;
    END;
    END;
    RETURN; END FL;
```

The temporary complexity of the procedure FL is equal to $O(N^3)$. Operation of procedure FL utilizes both the rows of the matrix (J loop) and the columns (I loops).

Due to the great thinness of the contiguity matrix (as well as the reachability matrix), particularly for a real semantic network, representing matrices A and D in computer memory as square arrays is not expedient. It is more promising to represent the matrices as contiguity lists[3]. Each point on the network corresponds to a contiguity list (for matrix D — a reachability list), i.e., a list of points contiguous with the given point (reachable from the given point for D).

If the matrices are represented in computer memory as lists, isolation of a matrix row is a simple task, whereas isolation of a column is a more complex process.

Therefore, the system utilizes another method to construct the reachability matrix from the contiguity matrix. The method is based on the following property of an oriented graph: Let $B(v)$ be the set of points reachable from point v, and suppose lines extend from point v to points $v_1, v_2, \ldots, v_m$; then

$$B(v) = \bigcup_{i=1}^{m} \{B(v_i) \cup v_i\}.$$

In other words, to determine the set of points reachable from a given point v we need but know the set of points reachable from those points to which there are lines from v. Since the points of the semantic network are topologically ordered, due to the property of topologic sorting the number of point v is less than the numbers of points reachable from v. Consequently, it is desirable to select points in the order which is the reverse of the topologic order. Therefore, when the next point is taken up for analysis, information is already available in the sets of points which can be reached from those connected by lines to the point being analyzed.

The procedure for constructing the reachability matrix D from contiguity matrix A based on this method is as follows:

```
SM: PROC(A, D, N);
    DCL (A(N, N), D(N, N)) BIN(1);
    D=A;
    DO K=N-2 TO 1 BY -1;
    DO J=K+1 TO N-1;
    IF A(K, J)=1 THEN
    DO I=J+1 TO N;
    IF D(J, I)=1 THEN D(K, I)=1;
    END;
    END;
    END;
    RETURN; END SM;
```

The temporary complexity of procedure SM, like that of procedure FL, is $O(N^3)$. However, in contrast to FL, as procedure SM operates only the rows are used (J loop and I loop), simplifying implementation of procedure SM when the matrices are stored as lists. For this reason, SM procedures are used in the ASAT-2 system to construct the distance and maximum path matrices.

Examples of analysis tasks. The third stage of operation of the system is performance of the analysis tasks listed above. The availability of the distance and maximum path matrices allows these tasks to be performed quite effectively.

Let us look at some examples of the use of the distance and maximum path matrices to perform some of these tasks.

One important task is determination of the presence and type of semantic connections between two terms. Three cases of relationships between terms must be considered:

— There is a semantic connection if one of the terms is a semantic component of the other or there is a third term which is a semantic component of the two terms;

— there is semantic conjunction when the terms are not semantically connected but there is a third term which is a semantic derivate of both the terms;

— there is neither semantic connections or semantic conjunction.

Let us assume that two terms are fixed by the topologic numbers I and J, where $I>J$. To perform the task, we utilize the distance matrix S. The task is performed by the following procedure:

```
REL: PROC(S, N, I, J);
     DCL S(N, N) BIN FIXED(8);
     IF S (J, I)>0 THEN
     PUT EDIT ('ТЕРМИНЫ СВЯЗАНЫ')(A(15));
     ELSE DO;
     DO K=1 TO J-1;
     IF S(K, I)>0 & S(K, J)>0 THEN DO;
```

```
PUT EDIT ('ТЕРМИНЫ СВЯЗАНЫ')(A(15));
RETURN; END;
END;
DO K=I+1 TO N;
IF S(I, K)>0 & S(J, K)>0 THEN DO;
PUT EDIT ('ТЕРМИНЫ СОПРЯЖЕНЫ') (A(17));
RETURN; END;
END;
PUT EDIT ('ТЕРМИНЫ НЕ СВЯЗАНЫ')(A(18));
END;
RETURN; END REL;
```

Procedure REL has a temporary complexity on the order of $O(N)$ for any pair of terms, allowing interactive operation with the system. In the ASAT system, this task executes in a period of time on the order of $O(2^N)$.

Another no less important analysis task is that of constructing the chain of maximum length connecting two terms. The task is performed using the maximum path matrix P:

```
DCP: PROC(P, N, I, J);
     DCL P(N, N) BIN FIXED(8);
     L=P(J, I); PUT LIST (J);
M:   IF L=1 THEN GO TO M1;
     L=L—1;
     DO K=J+1 TO I—1;
     IF P(J, K)=L THEN
     IF P(K, I)=1 THEN DO;
     PUT LIST(K); J=K; GO TO M; END;
     END;
M1:  PUT LIST(I);
     RETURN; END DCP;
```

The temporary complexity of procedure DCP is less than $O(N^2)$. In the ASAT system, this type of task is not performed due to its great complexity.

Similar procedures exist for performance of the other analysis tasks.

Implementation of the system. ASAT-2 is implemented in software written in *PL/1* and oriented toward the YeS series of computers.

A semantic network (as well as distance and maximum path matrices) are stored in computer memory as contiguity lists. A list element for point i contains the following information: j — the number of a point which can be reached from point i; RS — the length of the shortest path from point i to point j; RP — the length of the critical path from point i to point j; PP — a pointer to the next element on the list.

The operator which declares a list element is as follows:

```
DCL 1 ABB BASED(PQ),
      2 J BIN(16),
      2 RS BIN(8);
      2 RP BIN(8);
      2 PP PTR;
```

When a semantic network is represented in computer memory, the number of elements of type ABB is equal to the number of lines in the network. During the second stage of system operation, the distance and maximum path matrices are constructed. These matrices are assigned the variable RS and RP, respectively. This increases the number of list elements, since list elements are generated corresponding to paths leading to terms from indirect semantic components. However, experiments have shown that this increase is not significant, since the mean number of semantic components in the lexicon (determining the mean number of elements in a list) is just a few, much less than the total number of terms in the lexicon and, consequently, the use of the list form for matrices is justified.

The input of the system receives a list of terms. This list is prepared from the definitions of terms in the interpreting dictionary. A dictionary entry in an interpreting terminological dictionary is converted to a sequence of terms, the first of which is the term defined (it is marked with the symbol #), followed by terms from the definition, adjusted to a standard form (usually nominative singular; marked with the symbol H).

For example, the dictionary contains the following definition: "Segment — an entry in a hierarchical structure." This dictionary entry is transformed to the following sequence of terms:

# SEGMENT
H. ENTRY
H. HIERARCHICAL STRUCTURE

The system, as we have stated, operates in three stages. The first stage performs tasks from the second group, after which the user can perform a number of operations to edit the primary terminology, namely: Elimination of errors in entry of terms; changing of the composition of terms (in particular, cases may occur in which a term present as a direct semantic component is not one of the header terms; then, if this term belongs to the subject area in question, the user may decide to enter it into the dictionary as a header term); eliminate vicious circles, either by breaking one of the connections between terms or by changing the definition of one or more terms (the system can operate in a mode in which a double bond between terms such as A⇔B is automatically broken); movement of a term from the list of initial terms to the list of defined terms, or vice versa.

In the second stage of the system, this edited terminology is processed, constructing the matrices mentioned above.

In the third stage of operation upon user request, information is output as required for analysis, i.e., the tasks of groups one and four mentioned above are performed. Supplementary information (lists of terms sorted according to various characteristics; the total number of both direct and direct and indirect connections in the lexicon, etc.) is also generated, facilitating analysis of the terminological lexicon.

The ASAT-2 system was tested on a terminological dictionary "data bases in computer networks," containing 268 terms[6], as well as several fragments of terminological and general lexicons.

REFERENCES

1. Skorokhodko, E. F. Semanticheskiye seti i avtomaticheskaya obrabotka teksta (Semantic Networks and Automatic Text Processing). Kiev, Nauk. dumka Press, 1983, 220 pp.

2. Mamedova, M. G., Skorokhodko, E. F. Avtomatizirovannaya sistema analiza terminologicheskoy leksiki (Automated System for Analysis of a Terminological Lexicon). Scientific and Technical Information Series 2, 1981, No. 1, pp. 14-18.

3. Yevstigneyev, V. A. Primeneniye teorii grafov v programmirovanii (Use of Graph Theory In Programming). Moscow, Nauka Press, 1985, 352 pp.

4. Ore, O. Teoriya grafov (Graph Theory). Moscow, Nauka Press, 1980, 336 pp.

5. Reyngold, E., Nivergelt, Yu., Deo, N. Kombinatornye algoritmy - teoriya i praktika (Combinatorial Algorithms — Theory and Practice). Moscow, Mir Press, 1980, 476 pp.

6. Terminologicheskiy slovar po probleme "Bazy dannykh v setyakh EVM" (Terminological Dictionary On the Problem "Data Bases In Computer Networks") Edited by Yu. M. Gornostayev and L. N. Sumarokov. Moscow, International Center For Scientific and Technical Information, 1984, 76 pp.

UDC 681.3.02

Supplementary Facilities For Logical SM Computer Data Base Planning

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian (manuscript received 03 Sep 86; after revision 03 Aug 87) No 3, May–Jun 1988, pp. 64–65

[Article by I. A. Boychenko, O. M. Meshkov, A. V. Peshkov and I. I. Polonetskiy]

[Text] **Introduction.** Planning of data bases is a lengthy and cumbersome process. For data containing 500 or more elements, this process may take months, in some cases even years[1]. Basically, the data base planner is guided by his intuition and experience, and the quality of the results produced is quite doubtful. Poorly planned data bases hinder the process of application planning, increase the time and complexity of implementation of a data system and the time of development. In existing automated information systems based on DBMS, the creation of effective data bases is hindered by the following factor[1]:

— Insufficiently deep analysis of data requirements (including the semantics of names of attributes and interconnections among data items);

— the lengthy process of structuring, doubtful and difficult with manual processing;

— difficulties related to testing the planning process.

Elimination of these shortcomings requires supplementary facilities to automate the process of logical planning of data bases for the SM computer — the PROYEKT BD application program package.

The planning process. One effective method for development of logical data base models is the introduction of a number of relational data model concepts[2], reducing the data structure to third (or fourth) normal form. Structuring of data according to the third (fourth) normal form yields a clear representation of the subject area, precise definition of data base processing rules, simple data organization, decreasing the probability of performing incorrect operations[3].

The preparatory stage in logical data base planning is that of creating a conceptual model of the data as the complete set of all requirements for the data, obtained from user concepts of the real world and reflecting the data elements and their interconnections obtained as a result of collection and analysis of requirements for the data.

Individual local representations are input in multiple–terminal access mode and are transmitted to automated initial processing procedures, during which the local representations are edited and combined into a composite model.

Editing is an interactive process performed until all situations recorded by diagnostic messages have been resolved.

Based on the composite model generated, automated procedures of local structuring and creation of a canonical model are performed. The relations in fourth normal form obtained as a result of logical structuring represent a canonical logical data base model, an effective plan of the relational data base logical structure.

However, the planner himself can make changes to the canonical model. The principles for changing the canonical model may vary:

— To increase throughput;

— to adjust relationships in order to allow their use by individual application programs;

— in response to changing data requirements;

— in response to limitations on data structures in a specific DBMS.

The canonical model is changed by automated procedures for plan improvement.

Data base interactive planning language. This is the language by which the developer interacts with the automated system as required for entry of data requirements, editing and logical planning. This language is basically oriented toward the use of menus of functions to be performed. The hierarchical principle of construction of such menus and screen formatting is used. The creation of the language had two goals: Minimization of the number of characters input through the keyboard, simplicity and ease of understanding.

The first (starting) menu is used to select one of the basic functions:

— Entry and editing of local representations;

— creation of a canonical model;

— improvement of a structure;

— presentation of diagnostic reports;

— replanning of a structure.

Data entry is performed by filling out and editing forms on the video terminal screen. Data requirements are input on the basis of the local representation. The required field positioning and transition from one field to the next on the same row are performed automatically. Transitions upward and downward among rows on the screen are performed using the function keys.

Software. The controlling component of the PROYEKT BD software package is the DISPATCHER subsystem, which configures the package, creates the access file, and control the operation of subsystems. When a subsystem is called, control is transferred to the ACCESS subsystem, which verifies the access rights of the planners and permits or forbids access to the subsystem called.

The correspondent subsystem is used for entry and editing of local representations (full and abbreviated names of attributes, associations among attributes) in two special screen formats. Special screen formats are used to simplify entry of large local representations, menu mode is used for queries, a programmable full-screen editor is used to edit input data.

Composite and canonical models are created by the MODEL subsystem, which operates in menu mode and interacts with the planner using special screen formats.

The CENSOR subsystem is used to improve the canonical model generated.

The INTERFACE subsystem tests the limitations of the MIRIS DBMS, recodes attribute names from Russian to Latin characters and creates files describing schemas and subschemas ready for loading into the MIRIS DBMS in order to produce a plan of the data base logical structure.

The PLAN REPORTS subsystem documents the process of logical data base planning and prints out the following lists: Local representations, mismatched associations, excess connections, incomplete functional dependences, transitive connections, primary keys, candidate for secondary indexing, relationships, connections from compound keys to attributes of local representations, as well as charts of relationships, connections between attributes of local representations and distribution of attributes among relations.

To replan the logical structure of a data base, the RETURN subsystem makes the transition to any stage or procedure of planning, clearing the results of previous planning.

Conclusions. PROYEKT BD functions in the environment of OS RV 3.0 (or RSX 11M V.4.0) and the MIRIS (or KVANT) DBMs on SM-4 and SM-1420 computer systems, and places no limitations on their functional capabilities. The memory required is 24K words. Disk space required is 102K words.

Planning time for a data base of 400-500 elements is 4-6 hours.

PROYEKT BD has been introduced in organizations of the Ministry of Ferrous Metallurgy, Radio Industry Ministry, Food Industry Ministry, Medical Products Industry Ministry, Ministry of Agriculture, State Supply System, as well as institutes and enterprises of other ministries. The annual economic effect per installation is 20,470 rubles.

Delivery and installation of PROYEKT BD are performed by the Voronezh Inter-Industry Specialized Operating System and System Software Fund.

REFERENCES

1. Hubbard, J. Avtomatizirovannoye proyektirovaniye baz dannykh (Automatic Data Base Planning). Moscow, Mir Press, 1984, 296 pp.

2. Atre, Sh. Strukturnyy podkhod k organizatsii baz dannykh (Structured Approach To Data Base Organization). Moscow, Finansy i statistika Press, 1983, 215 pp.

3. Martin, J. Organizatsiya baz dannykh v vychislitelnykh sistemakh (Data Base Organization In Computer Systems). Moscow, Mir Press, 1980, 664 pp.

UDC 681.142:658.012.011.56(04)

Program System For Automatic Information Model Construction and Data Base Structure Planning

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian (manuscript received 27 Oct 86; after revision 02 Jun 87) No 3, May–Jun 1988, pp. 66–70

[Article by B. G. Nuraliyev]

[Text] A software system for the automation of planning of an information model and data base structure planning[1] was developed at the Moscow Economics-Statistics Institute in 1980–1985 on assignment from the USSR State Science and Technology Committee. Since 1986, the system has been maintained by the Centralized Library of Algorithms and Programs for Automated Management Systems, "Tsentrprogrammsistem" Scientific–Production Association, Ministry of Instrument Building, Automation Equipment and Control Systems (reg. No. 713). The system is designed for use with the SETOR DBMS and its analogues (SEDAN, SETKA, TOTAL), and runs under OS YeS, version 4.1 or later. This is the second package following the "Struktura" application software package[2] for automation of data base planning which has been distributed on a centralized basis in our country.

The system is designed to plan large integrated economic information systems. Particular attention was therefore given to accounting for the specifics of economic tasks such as: Large volumes of metadata; great specific share of bases in the set of requisite types; complex algorithmic interactions of characteristics with several levels of results; agreement of the names of the overwhelming majority of functional dependences of requisites with a predefined frame (pattern); extensive use of coding systems; absence of rigid boundaries between data and metadata; intersections of tasks in various functional services with respect to requisites and documents used; correlation of regulation processing with certain time periods; frequent variability of document form.

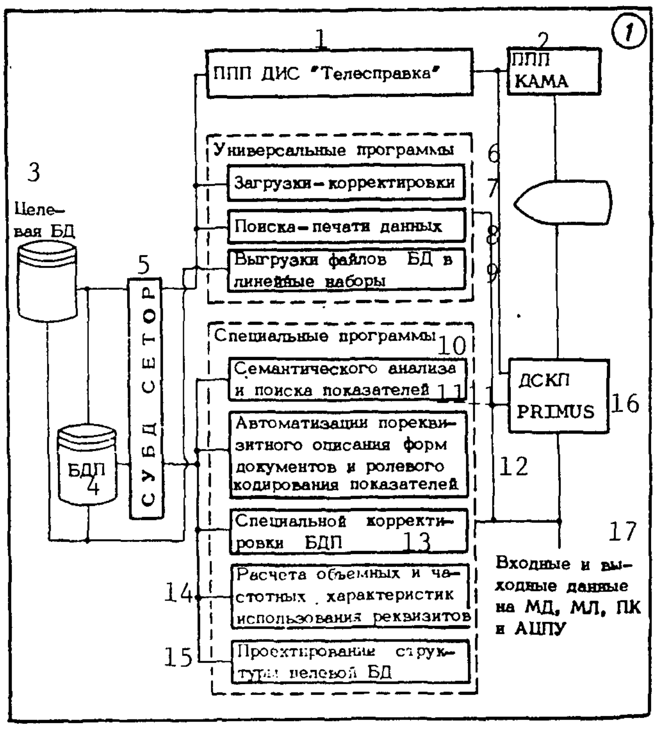
System architecture. The system consists of a set of programs (Figure 1), interrelated by an integrated planning data base. The SETOR network DBMS is used to maintain the planning data base. Figure 2 shows a fragment of the planning data base schema, the rectangles containing the SETOR master files, while the circles contain secondary files. The detailed planning data base solution is described in[3].

The system software consists of universal and specialized programs. Universal programs are used for loading and editing of the data base, retrieval and planning of data from the data base and unloading of data base files in linear sets. These programs are used both to manage the planning data base and to process any user data base in the SETOR or SEDAN DBMS; their advantages over the standard SETOR DBMS facilities include verification of correctness of data base processing, possibility of operation with files following an emergency (including restructuring and reorganization), highly effective access, and memory requirement, 80–120 Kbytes.

The system also includes special functional programs for semantic analysis and retrieval, automation of description of document forms by requisites and coding of characteristics, special planning data base editing, computation of frequency and volume characteristics of the use of requisites and characteristics as a function of their inclusion in document forms, and actual planning of the structure of a user data base.

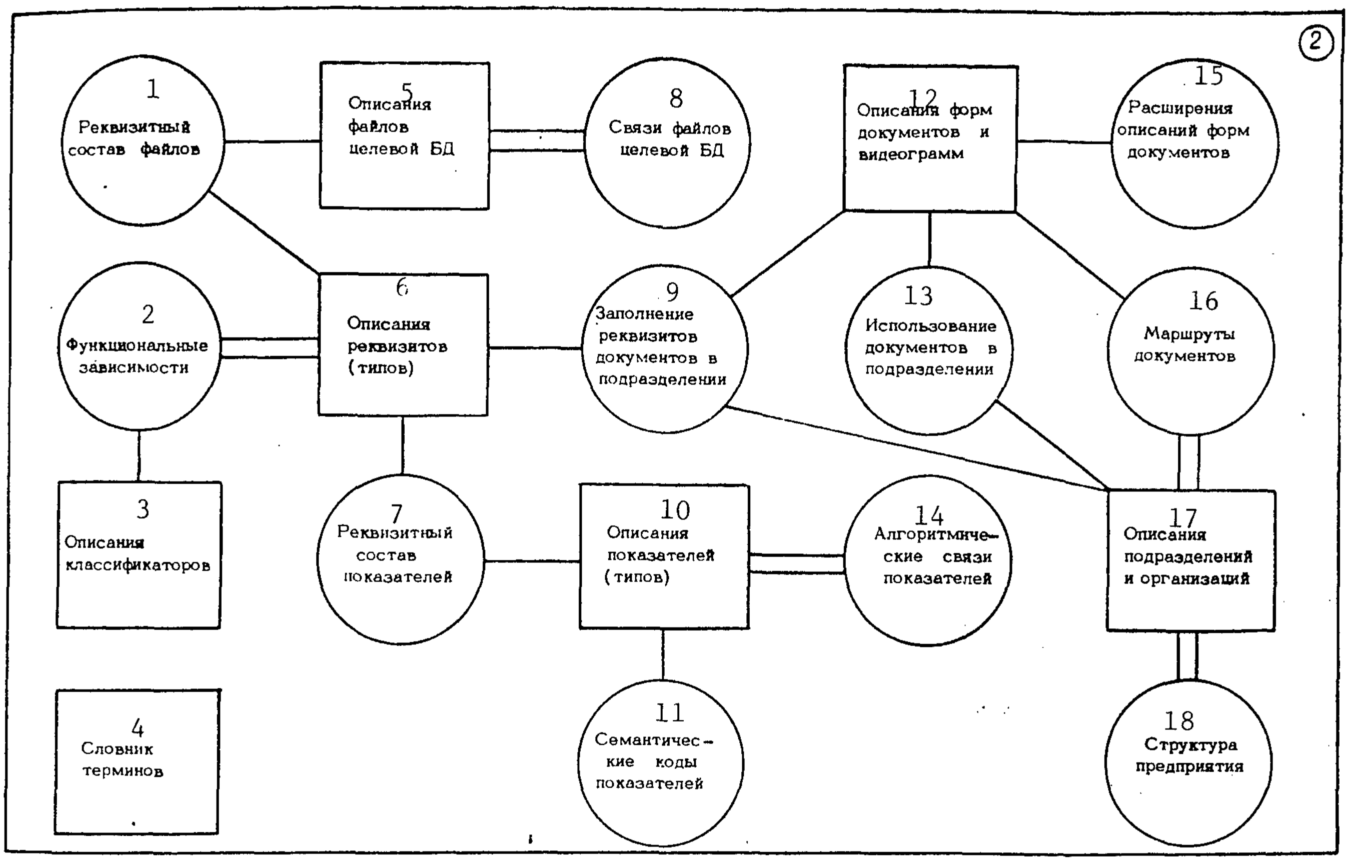
Interactive examination and editing of planning data base files are performed by the "remote reference" software package through the KAMA system or the PRIMUS dialog shared-use system (cf. Figure 1).

System usage techniques. Operations are started by formation and loading of preliminary versions of the term dictionary in the planning data base, list of classifiers, list of structural subdivisions and lists of the most frequently encountered requisites — object area characteristics.


1
ППП ДИС "Телесправка"
2
ППП КАМА
3
Целевая БД
БДП
4
5
СУБД СЕТОР
Универсальные программы
6
Загрузки-корректировки
7
Поиска-печати данных
8
Выгрузки файлов БД в линейные наборы
9
Специальные программы
10
Семантического анализа и поиска показателей
11
Автоматизации пореквизитного описания форм документов и ролевого кодирования показателей
12
Специальной корректировки БДП
13
Расчета объемных и частотных характеристик использования реквизитов
14
Проектирование структуры целевой БД
15
ДСКП PRIMUS
16
17
Входные и выходные данные на МД, МЛ, ПК и АЦПУ

Key: 1, "Remote information" software package; 2, KAMA software package; 3, user data base; 4, planning data base; 5, SETOR DBMS; 6, universal program; 7, loader–editors; 8, data retrieval and print out; 9, unload data base files to line sets; 10, special programs; 11, semantic analysis and search for characteristics; 12, automation of requisite description of document forms and role coding of characteristics; 13, special planning data base editing; 14, calculation of volume and frequency characteristics of requisite use; 15, planning of user data base structure; 16, PRIMUS dialogue shared-use system; 17, input and output data on magnetic disk, tape punch card and printer.

Subject area document and video screen formats, their requisites and characteristics are then described. Based on the experience of use of the system for planning of large economic information systems, the labor consumed in this stage represents up to 75% of the total labor consumed in the entire process[4]. Therefore, in contrast to most other methods, oriented toward the single user, we suggest parallel performance of operations in this stage by many workers with moderate qualifications. From the organizational standpoint, it is desirable to assign workers not to "tasks," but to specific subdivisions, subject area workers and document forms.


2
1
Реквизитный состав файлов
5
Описания файлов целевой БД
8
Связи файлов целевой БД
12
Описания форм документов и видеограмм
15
Расширения описаний форм документов
2
Функциональные зависимости
6
Описания реквизитов (типов)
9
Заполнение реквизитов документов в подразделении
13
Использование документов в подразделении
16
Маршруты документов
3
Описания классификаторов
7
Реквизитный состав показателей
10
Описания показателей (типов)
14
Алгоритмические связи показателей
17
Описания подразделений и организаций
4
Словник терминов
11
Семантические коды показателей
18
Структура предприятия

Key: 1, Requisite composition of files; 2, functional dependences; 3, classifier descriptions; 4, terms dictionary; 5, description of user data base files; 6, description of requisites (types); 7, requisite composition of characteristics; 8, user data base file connections; 9, filling of document requisites in subdivisions; 10, description of characteristics (types); 11, semantic codes of characteristics; 12, description of document and screen formats; 13, use of documents in subdivisions; 14, algorithmic connections among users; 15, expansion of document form descriptions; 16, document flow paths; 17, description of subdivisions and organizations; 18, enterprise structure.

The results of the examination are recorded on standard form. On each subject area document form a description blank is filled out, reflecting the name and code (number), nature and type, frequency and size characteristics of the document. Data are indicated on the movement of the document among subdivisions, methods and times of its preparation and use in the subdivisions, with informal comments by the planner.

Examination blanks are filled out for each requisite characteristic and each document format characteristic in the subject area. For a requisite characteristic, the code and name are recorded, plus the length in the given document, the use as a sorting key, inclusion in a single-line or multiple-line portion of the document, type of meaning and list of subdivisions filling in the requisite in the document. For a characteristic, the code and name (coinciding with the code and full name of the requisite base) are given, plus the length of the requisite base, composition of requisite characteristics, sequence of computation, list of subdivisions filling in the characteristic in the document, etc.

When examination blanks are filled in, particular attention is given to simplicity of filling in the documents. For example, in contrast to[2], the planner need not state the frequency and size characteristics for each characteristic and requisite: They are computed by system software based on the document characteristics. The mean time requirement for description of one document form is two hours plus 0.1 hour per requisite.

Data from the blanks are entered on machine-readable media and loaded into the planning data base by the universal loader-editor program.

To plan the combined economic information system data base, the results of the examination, obtained in parallel, are tied together into a common information model at the requisite level. The complexity of this operation consists in that the same characteristics (requisite bases) and requisite-characteristics have different names (synonyms) in different document forms and subdivisions. Cases of homonyms and polysemia are also encountered.

The information model stored in the planning data base is modified according to the results of the operation of formation of the correct identifiers. For example, if one of the synonym characteristics is used in two documents, while another is used in three, a single characteristic is selected for inclusion in all these documents.

A number of program checks for consistency of the information model are then performed, after which the planning data base is ready for machine planning of the user data base or for use as a reference dictionary for manual planning.

With machine planning, a program is run to compute the size and frequency characteristics of the use of characteristics and requisites based on the current planning data base status.

The facilities of the system utilize the algorithm developed for automatic planning of the user system information base, consisting of the integrated master and secondary files (supported by the SETOR, SEDAN, SETKA, TOTAL or SIOD network DBMS), as well as sequential access files. The input of the algorithm is the information model stored in the planning data base. The user has the opportunity to edit the initial and/or output data generated in planning in the planning data base.

It is possible to use the system in the stage of maintenance of the software. Its program facilities allow partial replanning of a user data base. The use of the planning data base as a dictionary system, in addition to the more common capabilities, supports the retrieval of metadata by formatting a characteristic name example, simultaneously storing several versions of the plan and integrating plan and user data.

Metadata identification methods. The following methods of identifying characteristic names are known[5]: Key word in context identifiers with indexes, categorizing of requisites and stylized names. These methods are similar in terms of dependence of labor requirement with the number of names analyzed to pairwise comparison of all characteristics. Therefore, for the economic information system of a middle-sized industrial enterprise, they require excessive labor expenditures.

In connection with this, a method has been developed for the system for unique identification of characteristics by automated semantic analysis of their names. The names of characteristics are described using a simplified role (frame) approach: The sense of a name is unambiguously coded by an indication of "performer in role," i.e., answers to fixed questions (concerning the essence, details, time and other characteristics) in terms of the economic subset of the natural language. A traditional seven-role frame[6] and/or simplified five-role frame may be used. A dictionary of terms is used in which each term corresponds to a code, with synonyms having identical codes. Coding of terms is performed either by the program or manually.

After loading of the results of examination into the planning data base, a program is run to analyze the semantic similarity of characteristic names. It utilizes two cluster functions for the "distance" between the names of characteristics j and k:

$$K_{jk} = \sum_{i=1}^{m} n_{ijk} Q_i;$$

$$K_{jk} = \sum_{i=1}^{m} Q_i \frac{2n_{ijk}}{l_{ij} + l_{ik}},$$

where n_{ijk} is the set of differences between terms in role i, i.e., terms present only in code j or code k; Q_i is the penalty for one difference in role i; m is the number of role names; l_{ij}, l_{ik} represent the number of terms in role i with semantic code j or k, respectively.

Semantic codes are considered similar in $K_{jk} \leqslant K_{\max}$. The "distance" functions meet the strict requirements of cluster analysis: $K_{ab} \geqslant 0 \forall a, b \in P$, where P is the set if characteristics analyzed; $K_{ab} = 0$ when and only when the semantic codes a and b coincide completely; $K_{ab} = K_{ba} \forall a, b \in P$; $K_{ab} \leqslant K_{ac} + K_{cb} \forall a, b, c \in P$.

Proper selection of the parameters $K_{\max}$ and Q_i, $i = \overline{1, m}$, with completeness of output at least 0.97 provides level of relevance 0.2–0.3; on the average, a planner manually selects one synonym for each 4–5 selected by the program as similar in meaning.

We note that for the method of semantic analysis suggested the number of manual comparisons of names S is proportional to the actual number of synonyms C_f: $S = \frac{1}{R} C_f$ (where $R \approx$const is the level of relevance), whereas for the methods listed above it is proportional to the square of the number of characteristic names[5]. Therefore, in spite of the greater machine time cost[4], the use of the method suggested is expedient. Thus, for 3153 characteristics in the accounting system of the "Frezer" plant, for the method of semantic analysis S=5563, 2–4 orders of magnitude lower than for other methods.

Since the number of requisite-characteristic types is usually significantly less than the number of requisite-bases, while the lengths of characteristic names average 2–3 times, we suggest a two-stage method for their unique identification: Categorization with subsequent analysis of "key words in context" within each category. Thus, in the accounting system of the "Frezer" plant, 122 requisite-characteristics were found (not counting document numbers); following division into six categories: Codes, numbers, signatures, dates, names, other S=299.

User data base planning algorithm. One peculiarity of the algorithm is the combination of analysis of functional dependence of requisites (during examination they are reflected in the characteristics, coding systems, etc.) with automated separation of the essence of the subject areas from the results of the examination. The algorithm includes the following main steps.

1. Determination of the list of data base master files.

1.1. Each requisite–characteristic x_j for which the total number of document lines sorted by x_j and one-time queries for x_j per unit time $E(x_j) \geqslant E_n$, is interpreted as key k_i of master file m_i.

1.2. For V $x_0 \in \{k_i\}$, the functional dependences are analyzed: $x_0 = f(x_1, \ldots, x_h)$ ($x_1, \ldots, x_h$ are the requisite arguments, the values of which unambiguously define the function x_0).

1.2.1. For V k_i, the existence of "one to one" (1:1) relations is verified, i.e., such that $k_t = f_1(x_v)$ & $x_v = f_2(k_t)$. A list of requisites is generated related to each other as 1:1:$\{k_t, x^1_v, x^2_v, \ldots\}$. From it descriptor x^{d}_v is selected: $L(x^{\mathrm{d}}_v = \min\{L(x^i_v), L(k_t)\}$V $x^i_v \in \{k_i\}$, where $L(x)$ is the length of requisite x. We note that $\{k_t, x^1_v, x^2_v, \ldots\} \cap \{k_i\} \neq \varnothing$, since $k_t \in \{k_i\}$. Where $L(x^\xi_v) = L(x^\eta_v)$, $x^\xi_v, x^\eta_v \in \{k_i\}$, if $E(x^\xi_v) < E(x^\eta_v)$ instead of $\{k_t, x^\xi_v, x^\eta_v, \ldots\}$ we study $\{k_t, x^\eta_v, \ldots\}$. The relationship of "being a descriptor" D is fixed (only for requisites for which a descriptor was not previously assigned) $x^{\mathrm{d}}_v = D(x^i_v)$:V $x^i_v \in \{k_t, x^1_v, \ldots\}$, such that $x^i_v \neq x^{\mathrm{d}}_v$ and $x_j \neq D(x^i_v)$V x_j. If a certain descriptor was previously defined for k_t: $x_{\mathrm{p}} = D(k_t)$, then for V $x^i_{\mathrm{n}} \in \{x^i_v\}$, $x_j \neq D(x^i_{\mathrm{n}})$V x_j, $x_{\mathrm{p}} = D(x^i_{\mathrm{n}})$ is fixed.

1.2.2. The 1:1 situation between keys of master files $k_t = D(k_v)$ is interpreted as a name code. A unique non-key "code" field $a_{v_1} = k_t$ is entered in a file with key name m_v ($\{a_{ij}\}$ is the set of non-key requisites m_i). All connections are closed in a file with key code m_t. In case of access through k_v, the reference file m_v supplies code k_t and the connection of the object will be selected by examining chains based on key k_t from m_t.

1.2.3. "One to many" (1:M) situations are interpreted between the keys of the master files: $k_t = f_1(k_v)$ & $k_v \neq f_2(k_t)$. A dependent file d_q is formed, related to the master file $m^*_{q1} = m_v$ and $m^*_{q2} = m_t$ by connecting keys $k^*_{q1} = k_v$ and $k^*_{q2} = k_t$, respectively. The set of non-key requisites of the file consists of arguments of functional dependences $\{e_{q0}\} = \varnothing$ and the set of non-key function requisites $\{b_{qs}\} = \varnothing$.

2. Sequential placement of requisite characteristics which are not keys of the master files $x_0 \notin \{k_i\}$.

2.1. If $\exists x_d = D(x_0)$, then $a_{d,\mathrm{cur}} = x_0$, the dependence $x_0 = f(x_d)$ is not studied, in all remaining dependences $(x_0 = f_\gamma(\{x_i\}))$ is replaced by $(x_d = f_\gamma(\{x_i\}))$V $x_i \neq x_d$. If this generates new, previously undescribed dependences x_d, they are processed as in 1.2.

2.2. If $x_d \neq D(x_0)$V $x_d \in \{k_i\}$, the functional dependences of x_0 are analyzed.

2.2.1. In place of the arguments of the dependence where possible their descriptors are substituted: If $x_0 = f(x^1_i, \ldots, x_i^{c-1}, x_i^c, x_i^{c+1}, \ldots) \exists x^c_i{:}\exists x_d = D(x^c_i)$, then $f(x^1_i, \ldots, x^{c-1}_i, x_i^c, x_i^{c+1}, \ldots)$ is replaced by $d(x^1_i, \ldots, x_i^{c-1}, x_d, x_i^{c+1}, \ldots)$.

2.2.2. Typical contradictions among functional dependences of the requisite x_0 are eliminated: $A \subset \{x_i\}$, $x_0=f(A)$, $B \subset \{x_i\}$, $x_0=f(B)$, $A \subset B$ (including $A=B$), then the dependence $x_0=f(B)$ is not analyzed. If $x_0=f(x_1, \ldots, x_\beta)$, $x_0=f(Q)$, $x_1=f(Q)$, $\ldots$, $x_\beta=f(Q)$, $Q \subset \{x_i\}$, the dependence $x_0=f(Q)$ is not analyzed.

2.2.3. Dependences x_0 are placed in sequence in the data base.

2.2.3.1. If $x_0=f(k_i)$, then $a_{i,\text{cur}}=x_0$.

2.2.3.2. If $x_0=f(x_1, \ldots, x_g, x_{g+1}, \ldots, x_h)$ (after eliminating 1:1, 1:M among the arguments), $x_1, \ldots, x_g \in \{k_i\}$, $x_{g+1}, \ldots, x_h \notin \{k_i\} \{x_1, \ldots, x_g\} \neq \varnothing$, the we select d_q: $\{k^*_{qr}\}=\{x_1, \ldots, x_g\}$, $\{e_{q0}\}=\{x_{g+1}, \ldots, x_h\}$ and $b_{q,\text{cur}}=x_0$. If d_q is not found, it is generated: $b_{q1}=x_0$. In this step, multiple connections may arise between the file pair d_q and m_p differing in the names of the connection l_{qr}

2.2.3.3. If $\{x_1, \ldots, x_g\}=\varnothing$, we seek or form an isolated line set w_i for which the set of requisite-arguments $\{u_{is}\}=\{x_{g+1}, \ldots, x_h\}$, the current requisite function $C_{i,\text{cur}}=x_0$ (for the newly generated file $C_{i,1}=x_0$).

3. Sequential placement of characteristics $x_0=f(x_1, \ldots, x_h)$.

3.1. Analysis of algorithmic connections of characteristics: If $(\Pi(x_0)=\Pi(Д\alpha)$ for $\forall\ Д\alpha \in V$, such that $x_0 \in \{x_i\}^{Д\alpha}) \,\&\, (\forall\, x_2 \neq x_0 : @((x_2=I(x_0)) \,\&\, (\Pi(x_2) < \Pi(x_0)))$, the characteristic $x_0=f(x_1, \ldots, x_h)$ is not analyzed, where $Д\alpha$ is the symbol of a document; V is the set of input documents $\{x_i\}^{Д\alpha}$ is the set of requisites of document $Д\alpha$; $\Pi(x_i)$ is the frequency of computation of characteristic x_i; $\Pi(Д\alpha)$ is the frequency of generation of document $Д\alpha$; $x_2=I(x_1)$ — characteristic x_1 is the algorithmic input for x_2.

3.2. The remaining characteristics are placed in the data base as in 2.2.1. and 2.2.3.

We have analyzed the experience of use of the system for planning of several actual data bases. In particular, the same examination results for an automated management subsystem developed by an enterprise were processed by several completing planning algorithms: The "Struktura" application software package[2], the "essence-property-relation" method and the DBDA software package, adapted for network structures[7].

Comparison of the parameters of the plans produced showed that the orientation of the algorithm suggested toward economic projects assures effectiveness of the user data base generated[8]. Thus, consideration of the algorithmic connections among characteristics, for example storage of running totals instead of primary characteristics, allowed a sharp reduction in data base volume. The use of information on functional dependences, frequency and size characteristics of metadata, the orientation toward partial use of classifier code (cf. 1.2.2.), the high degree of interconnection of the planned data base, allowing selection of the shortest connection chain to implement multiple-key queries, and other heuristics reduced program running time. At the same time, automatic isolation of subject area identifiers from the results of mass examination (cf. 1.1.) assured good data base structural stability.

The concept and experience of application of this system are presently being used to develop the COBA system, designed to automate examination and planning of data base structures for a relational DBMS on a 16-bit microcomputer.

REFERENCES

1. Sistema agregiruyemykh sredstv proyektirovshchika ASU (System of Combined Automated Management System Planning Facilities), Programmnyy kompleks avtomatizatsii postroyeniya informatsionnoy modeli i proyektirovaniya logicheskoy struktury bazy dannykh. Opisaniye primeneniya (Program System For Automating Construction of an Information Model and Planning of Data Base Logic Structure. Description of Application) Moscow, MESI Press, 1985, 36 pp.

2. Sistema avtomatizirovannogo postroyeniya struktur baz dannykh s ispolzovaniyem sredstv teleobrabotki — "Struktura" (The "Struktura" System For Automated Construction of Data Base Structures Using Remote-Processing Facilities). Kalinin: Tsentrprogrammsistem Press, 1981, 43 pp.

3. Nuraliyev, B. G. "Development of Metadata Base For Automated Management System Planning Automation System." Issledovaniye effektivnosti tekhnologicheskikh protsessov mashinnoy obrabotki dannykh (Studying The Effectiveness of Technological Processes In Machine Data Processing). Moscow, MESI Press, 1983, pp. 76-86.

4. Nuraliyev, B. G. "Automating the Correlation of Results of Operation of Individual Users In Examination of a Subject Area." Analiz effektivnosti i kachestva proyektirovaniya i funktsionirovaniya ASU v narodnom khozyaystve (Analyzing Effectiveness and Quality of Planning and Functioning of Automated Management Systems In the Economy). Moscow, MESI Press, 1983, pp. 138-139.

5. Nuraliyev, B. G. "Comparing the Effectiveness of Methods for Unique Identification of Characteristics and Requisites." Metody i sredstva povysheniya effektivnosti rabot po sozdaniyu sistem obrabotki ekonomicheskoy informatsii na EVM (Methods and Means of Increasing the Effectiveness of Work Related to the Creation of Computerized Economic Information Processing Systems). Moscow, MESI Press, 1985, pp. 22-30.

6. Yasin, Ye. G. "Theoretical Problems of Information System Development." Modeli dannykh i sistemy baz dannykh (Models of Data and Data Base Systems). Moscow, Nauka Press, 1979, pp. 5-30.

7. Hubbard, J. Avtomatizirovannoye proyektirovaniye baz dannykh (Automated Data Base Planning). Moscow, Mir Press, 1984, 296 pp.

8. Nuraliyev, B. G. "Estimating the Effectiveness of Methods of Data Base Planning On the Example of Development of a System For Processing User Plan Reports." Metodologicheskiye voprosy proyektirovaniya sistem MOEI (Methodological Problems of Planning of Queuing Systems). Moscow, MESI Press, 1985, pp. 12-23.

UDC 681.3.062

One Method of Synthesizing Virtual Relations

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian (manuscript received 25 Mar 87; after revision 03 Aug 87) No 3, May–Jun 1988, pp. 70–75

[Article by Yu. N. Radko]

[Text] Future development of the theory and practice of creating databases must be directed toward increasing the logical independence of the data stored and improving the "intelligence" of modules interpreting user queries[1]. Database management systems (DBMS) are equipped for this purpose with facilities to describe subschemas or "external representations,"[2] and smart data access systems are under development[3]. For databases using the relational data model[4, 5], studies are underway to create virtual relation synthesis operators[6, 7]. The principal distinction of such operators from database subschemas is that they are intended to provide an "external representation" to the user regardless of the current status of the database schema. If, as the information system is perfected and evolves, the database schema is altered, the corresponding subschema descriptions are also altered along with, possibly, some application programs. In this case, the subschema no longer provides full logical independence of database applications. Ideal virtual relation generating operators should be free of this shortcoming, since they are to provide in some sense "automatic" construction of a relational query formula and, therefore, must be tuned to the specific status of the database schema.

This article describes one method of creating an operator to read virtual relations, as well as its practical implementation in the language of relational calculus for the RINES relational engine[8].

The method proposed for the synthesis of virtual relations is based on the properties of areas of definition of relations in the basic relational data model. It is assumed that the names of attributes are unique within the limits of the database schema, reflecting their unique "role"[6]. A sufficient characteristic of any virtual relation is then its schema — the list and sequence of attributes indicated. Furthermore, attention is accented on one important property of the relational data model — it presumes storage in the database of the connections, i.e., copies of relations are not attribute-oriented. Permutation of attributes in the relational data model should be independent of the representation of the data stored, although it is not necessary, though desirable) that the database contain all permutations of relations[4]. Effective implementation of this method requires that the concept of the connection and relation not be differentiated as the information is stored and that for a relation of order N its $N!$ inversions (both of the schema and of its example) be stored, obtained in permutation of the attributes. This condition is satisfied by the RINES engine.

If the data base has connection, any attribute or group of attributes can equal probability be used as a search key, while any projection of a connection acquires the status of an independent relation for reading[9, 10]. These relations will be called homogeneous. For example, from a third power connection we can read (considering unary and binary): $3!/(3-1)!+3!/(3-2)!+3!/0!=15$ homogeneous relations. The characteristics of a homogeneous relation, as for a virtual relation, include the composition and sequence of its attributes.

Generally, the required virtual relation can be generated by a certain combination of homogeneous relations, related by natural connection[6, 7, 11-13]. Ideally, it degenerates to one homogeneous relation.

Let us study an example. Suppose a database consists of a set of normalized relations composed of the attributes: *A, B, C, D, E* and *F*. It is diagramed:

$$R1(\#A, B),\ R2(\#A, C),\ R3(\#A, D),$$
$$R4(\#C, B),\ R5(\#C, \#D, E),\ R6(B, C, F).$$

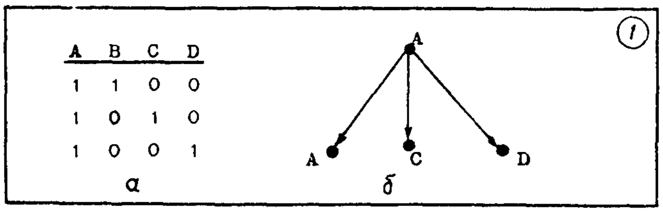
The symbol "#" us attached to attributes from the semantic "possible switch." We note that the coverage of functional dependences in this database is not minimal[12]. Let us study a certain virtual relation *RV*(*A, B, C, D*), It can be obtained by joining relations *R*1, *R*2 and *R*3. Studying the composition of the database, we can also note other combinations of relations generating *RV*.

Consequently, the virtual relation synthesis module must perform three tasks:

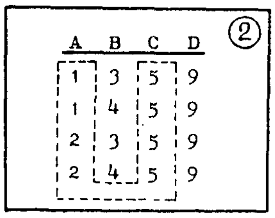
1) Determination of a certain optimal combination of homogeneous relations;

2) checking the combination obtained for correctness of the joining operation;

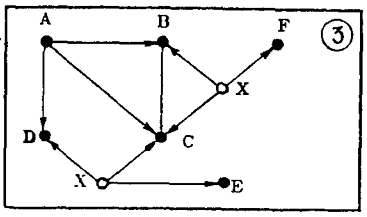
3) checking the connectedness of the combination and connecting homogeneous relations.



A	B	C	D
1	1	0	0
1	0	1	0
1	0	0	1



A	B	C	D
1	3	5	9
1	4	5	9
2	3	5	9
2	4	5	9



Connection matrix. The first task is performed using the concept of the connection matrix. The columns of the connection matrix are the names of the attributes in a virtual relation, placed in an assigned order (Figure 1). The matrix consists of zeros and ones. The columns of the connection matrix reflect the homogeneous relations selected to make up the virtual relation: If an attribute participates in a relation, a one is placed in its position, otherwise a zero. From the standpoint of combinatorial mathematics, a connection matrix is a (0, 1)-matrix of the incidence of the selected configuration of homogeneous relations in the set of virtual relation attributes.

The connection matrix is used (second part of third task) to perform the connection operation. The following algorithm can be suggested for this purpose, using train processing operators.

We represent: I — the number of the current homogeneous relation in the connection matrix from the top down, IM — the number of the next relation in the connection matrix, IK — the number of the current train for the Ith homogeneous relation.

Step 1. Setting of initial conditions. For all $I=1, \ldots, IM$, we set $IK=0$. The current homogeneous relation: $I=1$.

Step 2. For all homogeneous relations with numbers from I to IM (loop $I=I+1$) we select the next train $IK+1$ considering its connection with the previous higher homogeneous relations. If there is none or $I>IM$, go to step 3.

Step 3. $I=I-1$. If $I=0$, go to END. Otherwise, step 2.

If we consider the information processing quantum in this algorithm to be the selection of a train, its effectiveness in the sense of processing relevant information depends on the duplication of the trains. Then, if in the process of execution of the connection matrix for each homogeneous relation a unique, nonrepeating train is always selected, the effectiveness of virtual relation synthesis can be said to be 100%. Conversely, the connection matrix must be considered potentially redundant if the algorithm, when modeled in the general area of definition of the virtual relation, does not yield 100% effectiveness.

For example, suppose a connection matrix consisting of $R1$, $R2$, $R3$ is generated for an RV (cf. Figure 1), while the domains of the attributes $A=\{1,2\}$, $B=\{3,4\}$, $C=\{5\}$, $D=\{9\}$. The general area of definition of RV is shown in Figure 2. Clearly, the subset of possible trains is determined by the structure of the functional dependences. The dotted line shows trains which could be selected in relation AC without considering the functional dependences. They are duplicated, meaning that the connection matrix is potentially redundant.

According to the algorithm, the source of duplication of trains of the current homogeneous relation in a connection matrix may be any attributes from relations located in the connection matrix higher than the current relation, but absent in the current relation. For relation AC this would be attribute B. This condition is not met for attribute D, and therefore it does not influence the effectiveness of synthesis, which can be easily seen by changing domain D.

The set of attributes active in homogeneous relations above the current relation is called the set of known attributes, which we shall designate AI, while the set of attributes in the current relation will be designated AT.

Redundancy potential. We shall refer to the number of redundant duplicating trains for a certain homogeneous relation from a connection matrix obtained by repeatedly accessing the database as a potentially redundant homogeneous relation and represent it as U. Thus, for relation AC in the example $U=2$. The value of U depends on the area of the virtual relation attribute domains. And if in the example the areas of the domains are represented as M_A, M_B and M_C, respectively, for relation AC the redundancy potential

$$U_{AC}=M_A\cdot M_C\cdot M_C-M_A\cdot M_C=Q_{ABC}-Q_{AC}.$$

The quantity Q is called the scalar value of the dependency vector of the corresponding virtual relation attributes. The direction of the dependency vector is taken from the first left attribute in the virtual relation diagram to the right, since it is determined by the order of the attributes in the cartesian product which gives its area of definition. The general formula to compute the redundancy potential of the current homogeneous relation is:

$$U=Q_{\mathrm{AI}\vee\mathrm{AT}}-Q_{\mathrm{AT}}=Q_{\mathrm{AT}}\cdot(Q_{\mathrm{AI}\setminus\mathrm{AT}}-1). \quad (1)$$

Since Q is computed through the domain thickness, in order to estimate the redundancy potential of a relational DBMS, its linguistic facilities should include the function of determining the current number of elements in a domain.

Since the redundancy potential and, therefore, the area of definition of homogeneous relations from a connection matrix influences the general set AI10AT of attributes, when we represent the plan of homogeneous relations we must consider all attributes of the set, representing the location of attributes from the set AI\AT. For example, for homogeneous relations according to Figure 1 we can write: $R21$(A.C), $R31(A..D)$.

Attributes belonging to AT will be called useful attributes, those represented by the dots will be called parasitic attributes. The parasitic attributes to the left of the leftmost useful attribute will be called left insignificant attributes, while those written to the right of the rightmost useful attributes will be called right insignificant attributes.

Useful attributes are marked in the rows of the connection matrix by ones. We note that if the plan of a homogeneous relation contains no parasitic attributes, its redundancy potential is equal to zero.

First connection matrix optimality criterion. Just as one of the goals of normalization of a relation is to decrease potentially duplicated information[5, 11], let us select as the first optimality criterion of a connection matrix minimization of its summary redundancy potential for all rows:

$$U^{\mathrm{opt}}=\min\sum_{i} U_{i}. \quad (2)$$

In the ideal case, $U^{\mathrm{opt}}=0$. According to (1), this is possible, for example, if AI=Ø or the connection matrix is constructed of a single homogeneous relation.

However, the use of criterion (2) does not always yield an unambiguous result. This is because the redundancy potential is not an exhaustive characteristic of the properties of a homogeneous relation. Thus,

$$U_{.BC}=M_{A}\cdot M_{B}\cdot M_{C}-M_{B}\cdot M_{C},$$
$$U_{A.C}=M_{A}\cdot M_{B}\cdot M_{C}-M_{A}\cdot M_{C},$$

if $M_{A}=M_{B}$, then $U_{.BC}=U_{A.C}$. Therefore, it has been suggested that yet another characteristic of the homogeneous relation be studied — the strength of the connection between its attributes.

In the semantic interpretation, the strength of connection is an indication of the "dependence" between useful attributes in a homogeneous relation and determines the "rate" of change of values of write attributes with respect to left attributes in its area of definition. The strength of connection between two attributes X and Y is proportional to the number of changes of attribute Y with respect to a single change in attribute X. The force of connection is numerically equal to the specific portion of nonduplicated trains in the area of definition of the homogeneous relation without considering the insignificant, parasitic attributes. If I is the force of connection, by definition: $I_{.X.Y.Z.} = Q_{XYZ}/Q_{X.Y.Z} = Q_{AT}/V$.

The quantity V is called the volume of the area of definition of a homogeneous relation. For example, $I_{.BC}=1$, $I_{A.C}=1/M_B$.

Based on the force of connection with identical redundancy potential, we shall consider a homogeneous relation with greater force of connection to be preferable. The maximum force of connection is equal to one, the minimum does not reach zero due to the finite nature of any domain.

General connection matrix optimality criterion. Let us combine the two characteristics of a homogeneous relation into one. For each row of the connection matrix we introduce the quantity

$$R = UJ/I = V\cdot(Q_{\mathrm{AI}}\backslash_{\mathrm{AT}}-1),$$

where R is called the resistance of the homogeneous relation. Then the general optimality criterion of the connection matrix is:

$$\min\sum_i R_i \rightarrow \mathrm{MS}^{\mathrm{opt}}. \tag{3}$$

In other words, the optimal combination of homogeneous relations to generate a virtual (optimal connection matrix) relation must be considered that for which the sum of resistances of the rows is minimal.

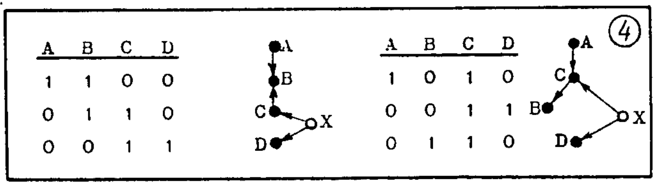
Let us study the problem of checking a connection matrix for correctness of the connection operation and for connectedness.

Graph of connection matrix dependences. Since a connection matrix is an incidence matrix of combinatorial configuration, it can be represented as a graph, the points of which are the attributes, while the lines are the rows of the connection matrix. We shall refer to such a graph as the dependence graph of a connection matrix or simply the graph of the connection matrix (cf. Figure 1). The graph of a connection matrix is an oriented graph. The orientation of its lines is selected in the direction of "functional dependence" between attributes. Thus, for line AB, direction from A to B means that the key in relation $R1$ is attribute A. The graph of a connection matrix may consist of points and hyperpoints. Let us present the full graph of dependences for our example database, constructed according to these requirements (Figure 3).

In the full graph, the artificially introduced points (attributes) X are represented as hyperpoints, representing group keys consisting of more than one attribute. A hyperpoint is always connected with its "daughter" points (attributes which make it up as a semantic key) by outgoing lines. Clearly, at least two lines always originate here.

The dependency graph of a connection matrix is used to check the correctness of construction of the connection matrix in terms of the connection operation. The checking method suggested is based on the theorem of the source of a connection result proven in the axioms of Armstrong[12]. In our case, the result of the connection is a virtual relation. We can affirm that the connection matrix is constructed correctly if it has a homogeneous relation such that its probable semantic key is a probable key for all virtual relation diagrams, i.e., the virtual relation functionally depends on this key. Such a key is called a source or root of the virtual relation.

Let us interpret this conclusion using the connection matrix graph. The connection matrix is constructed correctly in the sense of the connection operation if its graph contains a point or hyperpoint from which any other graph point (except the hyperpoint) can be reached in the direction of the "functional dependence" for the lines of the graph. For the graph of Figure 1, this point is that of attribute *A*. It follows from this that if the graph of a connection matrix can be represented as an oriented tree, the connection matrix is properly constructed. For comparison, Figure 4 shows improper connection matrices and their graphs which may be generated during *RV* synthesis.



A	B	C	D
1	1	0	0
0	1	1	0
0	0	1	1

A	B	C	D
1	0	1	0
0	0	1	1
0	1	1	0

4

In another statement, the task of seeking a correct and optimal connection matrix can be looked upon as the task of seeking an oriented partial dependency graph constructed in the set of attributes of the plan of a virtual relation. This article does not analyze the versions of application of known combinatorial algorithms to seek the minimum tree trunk[14].

The method we have suggested for testing a connection matrix for correct connections can also be applied to a connection matrix which has been constructed, as in the process of its creation, to select the next row. This allows immediate rejection of homogeneous relations which do not meet the condition of correct connection, and construction only of correct connection matrices. Obviously, the algorithm for selecting homogeneous relations in this case is recursive, which is confirmed by practical implementation.

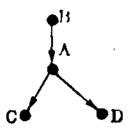
Synthesis of transitive dependence. A special case is possible in the synthesis of a virtual relation, in which the graph of the connection matrix is unconnected. For example, it is necessary to obtain the virtual relation *RV2* (*B*, *E*). Looking at the dependency graph of Figure 3, we see that there is no direct connection between attributes *B* and *E*. The connection matrix obtained will have an unconnected graph, or in other words will be undefined. To eliminate the uncertainty, it has been suggested that the plan of the assigned virtual relation be expanded by the inclusion of one or more transitive attributes. This expansion should be "invisible" to the user. The optimal transitive attribute can be selected using criterion (3). Thus, the virtual relation is converted to *RV2*(*B*, *C*, *E*) (cf. Figure 3). Detailed analysis of the algorithm used to search for optimal transitive attributes is rather complex and will not be described here.

Practical virtual relation synthesis operators may be limited as to the degree of uncertainty which can be eliminated from the connection matrix, i.e., the number of transitive attributes introduced to the connection matrix for one pair of unconnected attributes. Thus, the RINES relational converter[8] can eliminate first-order uncertainty in a connection matrix.

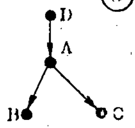
Synthesis time optimization. Suppose as a result of synthesis of a virtual relation a connection matrix is obtained (Figure 1). Let us now show that permutation of connection matrix columns (without disrupting optimality) can significantly influence the time required to execute the homogeneous relation connection algorithm. To do this, we introduce the concept of the connection matrix execution graph. Just as for the connections graph, in the execution graph the points are the attributes of the virtual relation, while the lines are the rows of the connection matrix. The difference is in the orientation of the line: For the execution graph they are not oriented in the direction of functional dependence, but rather in the direction of the dependence vector. Furthermore, the execution graph does not require the use of hyperpoints. For example, for the connection matrix of Figure 1 the execution graph corresponds to its dependency graph. Figure 5 shows execution graphs obtained by permutation of connection matrix columns (cf. Figure 1). Their lines ar oriented according to the changing direction of the dependency vector.

5

$$\begin{array}{cccc} B & A & C & D \\ \hline 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{array}$$



$$\begin{array}{cccc} D & A & B & C \\ \hline 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{array}$$



In order to estimate the connection time of homogeneous relations, let us introduce the concept of the moment of connection between a pair of attributes. For the pair *AB*, the moment of connection is defined as:

$$\mathbf{M}_{AB}=M_B/M_A.$$

Moment $\mathbf{M}_{AB}$ is equal to the number of trains in the area of definition of the relation *AB*, for which attribute *A* has the same value. Then the total number of database accesses require to extract *RV* according to the graph of Figure 1 can be estimated as:

$$\Sigma=M_A\cdot(\mathbf{M}_{AB}+\mathbf{M}_{AC}+\mathbf{M}_{AD})=M_B+M_C+M_D,$$

while the estimates of the number of database accesses for the graphs of Figure 5 are:

$$\Sigma=M_B\cdot\mathbf{M}_{BA}\cdot(\mathbf{M}_{AC}+\mathbf{M}_{AD})=M_C+M_D,$$
$$\Sigma=M_D\cdot\mathbf{M}_{DA}\cdot(\mathbf{M}_{AB}+\mathbf{M}_{AC})=M_B+M_C.$$

We can see from these expressions that the connection matrix of Figure 1 is clearly inferior in terms of execution effectiveness to its equivalents shown in Figure 5. This leads us to the conclusion: In a virtual relation, permutation of the columns of an optimal connection matrix determines the rate of information generation. The time can be estimated from the execution graphs by the use of the connection moments. This is clearly illustrated by analysis of any example of virtual relation *RV*.

It is interesting that computation of the connection moments leads us to some of the features of the query decomposition algorithm used in the *QUEL* language[11, 12].

Specifics of practical implementation. The method we have suggested for synthesis of virtual relations requires that the following steps be performed.

1. Set list and sequence of virtual relation attributes.

2. Find optimal (minimum resistance) and correct (in terms of connections) connection matrices.

3. Check connection matrix for connectedness of its graph, if necessary seek optimal transitive attributes. If transitive attributes cannot be defined, the virtual relation cannot be synthesized.

4. Optimize connection matrix execution algorithm by permutation of its columns, determine final form of connection matrix.

5. Produce virtual relation by connecting homogeneous relations with respect to connection matrix rows.

In practice, the method is implemented as an operator for reading of virtual trains, a supplement to the RINES language. One peculiarity of the databases used is that a full set of inversions for each loaded train is generated for each relation defined in the database. This assures connection in the database.

The operator is divided into two parts: The connection matrix synthesis module, the input of which receives a list of virtual relation attributes, and the virtual train generator module. The modules can be called in *PL/1* by means of the CALL operator, formatted by the preprocessor facilities. A virtual relation is generated by a single call to the connection matrix synthesis module, which creates it in the form of a bit matrix with a row length f 16 bits (maximum content of virtual relation up to 8 attributes). The virtual train generator module must then be called in the loop before the end train characteristic is reached, and sent to the entry of the connection matrix. As the loop executes, the value of the next virtual relation train will be selected from the database in *PL/1* variables.

The following simplifications of theoretical analysis were tested in the construction of the connection matrix synthesis module. The module was constructed on the assumption that the optimal connection matrix has the minimum number of rows. It can then be synthesized in a single pass if the following rule is used to select each subsequent connection matrix row: A row produces an optimal connection matrix if it has, first of all, the maximum number of "known" attributes — $\max Q_{AI \wedge AT}$, plus the maximum of "unknown" attributes — $\max Q_{AT \setminus AI}$, and does not violate correctness of connection of homogeneous relations.

With the additional condition of equality of all domain ranges, the connection matrix algorithm is made still simpler, while the recursive connection matrix synthesis module is decreased in size to 200–300 *PL/1* operators (not including database access operators). In spite of this, correct connection matrixs were generated in experimental operations for all of the requested virtual relations (over 100) in a commercial database with the following characteristics: Number of attributes — 140, connections (relations) — 153, of which 148 were binary, 5 were trinary.

It was noted that the effectiveness of generation of virtual relation was determined by the time required to select homogeneous relations from the database, in comparison with which the time required to synthesize the connection matrix was negligible.

The virtual train reading operator has been used in a universal multiterminal query system for RINES databases designed for untrained users. A table can be obtained by indicating the desired characteristics from a list of attributes. Certain conditions can be added for the characteristics, such as "equal to" or "greater than." The specific database status is considered during each connection matrix synthesis cycle.

REFERENCES

1. Sheldon, S. "Design of a Virtual Database." Information System, 1985, 10, No. 1, pp. 27-35.

2. Deyt, K. Vvedeniye v sistemy baz dannykh (Introduction to Database System). Moscow, Nauka Press, 1980, 464 pp.

3. Polyakov, V. I. "Smart Data Access Systems." USiM, 1983, No. 3, pp. 66-72.

4. Codd, E. F. "A Relational Model of Data For Large Shared Data Banks." Comm. ACM, 1970, 13, No. 6, pp. 377-387.

5. Dribas, V. P. Relyatsionnye modeli baz dannykh (Relational Database Models). Minsk, Bellorussian State University Press, 1982, 191 pp.

6. Maier, D., Rozenshtein, D., Warren, D. S. "Window Functions." Advances In Computing Research. London, YAI Press, 1986, No. 3, pp. 213-246.

7. Yannakakis, M. "Querying Weak Instances." Ibid, pp. 185-211.

8. Radko, Yu. N. "Hierarchical Structure For Modeling of Relational Tables." Programmirovaniye, 1987, No. 1, pp. 74-80.

9. Zaytsev, N. G. "Relational Homogeneously Structured Unconnected Database." USiM, 1982, No. 6, pp. 74-79.

10. Vasilev, A. V. "Development of Quasirelational Information-Retrieval Systems Using the SETOR DBMS." Tr. NII Radio, 1984, No. 3, pp. 110-115.

11. Ulman, J. Osnovy sistem baz dannykh (Database System Principles). Moscow, Finansy i statistika Press, 1983, 334 pp.

12. Kokoreva, L. V., Malashinin, I. I. Proyektirovaniye bankov dannykh (Planning of Data Banks). Moscow, Nauka Press, 1984, 256 pp.

13. Golosov, A. O. "Use of Classification Methods In Planning Relational Database Schema." Voprosy informatsionnoy tekhnologii (Problems of Information Technology). Moscow, VNIISI Press, 1982, pp. 63-67.

14. Papadimitriy, H., Steiglitz, K. Kombinatornaya optimizatsiya: algoritmy i slozhnost (Combinatorial Optimization: Algorithms and Complexity). Moscow, Mir Press, 1985, 512 pp.

UDC 681.14

Noise-Tolerant Digital Module Automated Printed Circuit Planning Subsystem

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian (manuscript received 16 Jul 86; after revision 10 Mar 87) No 3, May-Jun 1988, pp. 76-78

[Article by L. N. Kechiyev, S. N. Smirnov and I. V Tsirin]

[Text] **Introduction.** The constantly increasing speed of electronic equipment, decreasing levels of logical pulses, and increasing sensitivity of digital integrated microcircuits to electromagnetic interference have placed rigid requirements on the electrophysical parameters of printed circuit boards: Electrical capacitance, inductance, connecting line wave impedance, specific signal propagation delay in lines, effective dielectric permeability.

Digital integrated TTL microcircuits with Schottky diodes (TTLS) and emitter-coupled logic (ECL) are used to construct high-speed digital circuits, assuring switching times from one logic state to another of a few nanoseconds or less[1]. The development of scientific programs for the creation of superfast integrated circuits continues to decrease switching times[2]. Very fast digital modules based on these microcircuits can operate at clock frequencies of hundreds of megahertz, meaning that the transmission of logic signal leading and trailing edges over communications lines, including within printed circuit boards, requires transmission of frequencies of several hundreds of megahertz. For example, the leading edge of a 1 ns signal requires that communications lines carry spectral components at frequencies of up to 265 MHz. At such high frequencies, requiring preservation of wave impedance on printed-circuit communications lines, matched lines are desirable. Furthermore, as the speed of digital microcircuits increases, the permissible noise level at circuit inputs decreases. The amplitude of noise, called cross talk, is determined by mutual capacitive and inductive parameters of coupled lines, one acting as a source of noise for the other.

Particular attention must be given to the design of data exchange buses for units based on high-speed LSI microprocessor sets. Signal delays in individual bus lines must not exceed the permissible levels. Otherwise, operational failures may occur.

The following main tasks must therefore be undertaken in the design of printed circuit boards for noise-tolerant, high-speed digital modules.

1. Design of internal and mutual electrophysical parameters of connecting lines.

2. Calculation of propagation delays for signals and reflected noise in connecting lines, comparison of the values obtained with permissible values and determination of the need to match lines.

3. Calculation of cross talk levels and their minimization by design methods.

The solution of these problems, related to the problem of internal electromagnetic compatibility[3], must be produced in the board design stage, since the electrophysical parameters of connecting lines are determined by the geometric parameters of the printed circuit board after the layout stage is completed.

Known printed circuit board automatic design systems are intended to solve topologic problems of placement of elements and layout of connecting paths, with subsequent generation of design and technological documentation[4]. However, it must be considered more promising to create CADD systems permitting additional computation of the electrical parameters of boards and noise levels in them to allow timely modification of layouts if necessary. One example of this approach is the NESSY design system[5].

In this article, we present the results of development of a subsystem for automatic planning of printed circuit boards and modules considering internal electromagnetic compatibility (SAPR VEMS), which can meet digital module noise-tolerance requirements.

Design of the SAPR VEMS subsystem. The SAPR VEMS subsystem[6-8] is implemented using the ARM hardware system, based on the SM-4 microcomputer. The subsystem utilizes the standard shared-function real-time operating system RAFOS[9], the advantages of which include rapid reaction to external actions, simplicity and ease of use, high effectiveness and simple time sharing among several users, the ability to construct complete systems, consisting of RAFOS and application programs. The system can be operated by unsophisticated users.

The main programming language used in the system is FORTRAN. Interaction of application programs in the subsystem is organized using the PAGEN macroprocessor[9].

As programs in the subsystem run, information input by users is verified (syntax or data value check), and training-reference information is generated by the system of application programs.

Composition and organization of the SAPR VEMS subsystem. The SAPR VEMS subsystem is a system of application programs designed for engineering computation of electrical parameters of printed circuit units, distortion of signals in individual connecting lines on circuit boards, as well as the level of cross talk between two adjacent conductors. These calculations are related to the tradition tasks of topologic planning — layouts of elements and connection paths. A structural diagram of information exchange in SAPR VEMS is shown in the figure. The subsystem includes the following programs:

CAPCAL — calculation of capacitance parameters of printed circuits;

INCAL — calculation of inductive parameters of printed circuits;

RESCAL — calculation of connecting line wave impedances;

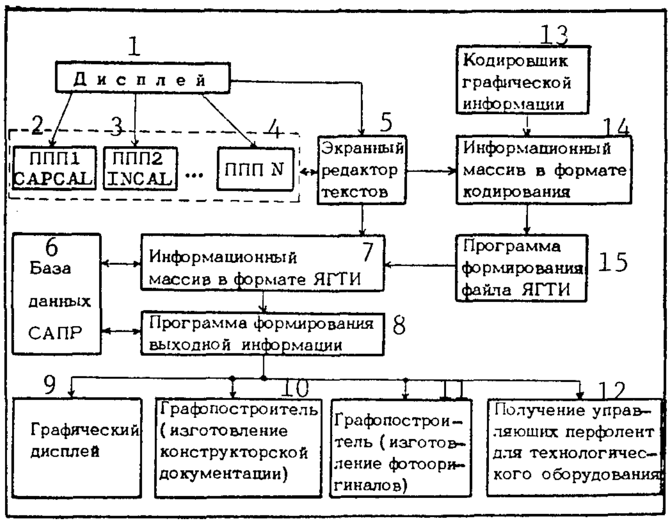
REFLEC — calculation of signal distortions in connecting lines due to reflection noise mismatch;

CROSS— calculation of cross talk noise levels.

SAPR VEMS software. Computation of capacitive parameters of a printed circuit unit is the major planning stage, since these parameters virtually determine the quality of functioning of a module.

During quasistatic analysis of connecting lines, the capacitive parameters of the line determine its inductive parameters and wave impedance, which, in turn, allows computation of the reflection noise and cross talk.

To main methods have been developed to compute printed circuit parameters: The combined methods of moments and boundary elements[10] and the modified finite elements method[11, 12].


1
Дисплей
2
3
4
ППП1 CAPCAL
ППП2 INCAL
...
ППП N
5
Экранный редактор текстов
13
Кодировщик графической информации
14
Информационный массив в формате кодирования
6
База данных САПР
7
Информационный массив в формате ЯГТИ
15
Программа формирования файла ЯГТИ
8
Программа формирования выходной информации
9
10
11
12
Графический дисплей
Графопостроитель (изготовление конструкторской документации)
Графопострои-тель (изготов-ление фотоори-гиналов)
Получение управ-ляющих перфолент для технологичес-кого оборудования

Key: 1, Display; 2, program 1 CAPCAL;3, program 2 INCAL; 4, program N; 5, full-screen text editor; 6, CADD database; 7, data file in YaGTI format; 8, output information formatting program; 9, graphic display; 10, plotter (preparation of design documentation); 11, plotter (preparation of photographic originals); 12, generation of punch tapes to control process equipment; 13, graphic information encoder; 14, data file in coding format; 15, YaGTI file formatting program.

The former of these methods is based on the indirect method of boundary elements, presuming the presence of a Green function for a heterogeneous unlimited area and discretization of the surfaces of the printed circuit conductors using the method of moments to describe the distribution of charges in it. Printed circuit board capacity parameters are calculated by approximating the surface of the conductors with elementary areas. The approximation of charges on the surfaces of these areas is selected depending on the placement density of conductors on the board and may be either a pulsed or stepped approximation.

The modified method of finite elements requires discretization of a certain volume containing the computation fragment of the printed circuit board into elements of finite dimensions. The Laplace equation for the electric field of the system of conductors is replaced by a system of algebraic equations allowing determination of the potential function at individual field points with fixed boundary conditions. The values of the potential function near the surfaces of electrodes can be used to determine the desired electrical parameter. Automation of all stages in the method allows it to be used practically without limitations as to environmental parameters and electrode configuration.

The inductive parameters of a printed circuit board can be determined either by conversion of the matrix of internal and mutual capacitances of the system of conductors or by using a method outlined in[13]. In this case, the printed conductors are approximated by a set of current-carrying filaments, for which the equations for internal and mutual inductance are known. Assuming current density through the cross section of the conductors to be constant, adding the inductive parameters of the filaments determines the inductive parameters of the printed conductors.

The reflection noise is calculated by the method of characteristics[14], in which computation of signal distortions using the concepts of dropping and reflected waves is reduced to analysis of linear algebraic equations. A cosine-line or spline approximation is used to describe the input and output volt-ampere characteristics of the digital integrated circuits required for computation, allowing the nonlinear nature of these characteristics to be considered.

The cross talk level is estimated for the beginning and end of a connecting line through the capacitive and inductive coupling coefficients[15].

Printed circuit board planning technology used in SAPR VEMS. The use of the SAPR VEMS subsystem requires that the placement of microcircuits and connecting path layout be completed by traditional methods. Subsequent printed circuit analysis is performed using the SAPR VEMS subsystem by computing the most "dangerous" board fragment, which is determined by analysis of the schematic diagram of the board. This fragment may be the longest signal conductor, for which the signal propagation delay or signal shape at the beginning and end of the line must be calculated. A long sector of conductors located close to each other can be analyzed to determine the cross talk level. In this case, if the wave impedance of a connecting line is known, computations can be performed during board design analysis.

Each program is narrowly specialized for use as a practical engineering method. The software package may include several programs implementing various computation methods. The user can select the most acceptable method based on the specifics of the task at hand.

The application program package includes a header module, task-oriented application modules and a control module.

The header module sends a message to the terminal concerning suggested calculation methods, a brief technical description of the methods and recommendations for the use of each method, then transfers control to the task-oriented module which is selected.

The task-oriented modules perform calculations by the method selected. Orderly input of initial data is supported, communications are organized with the system data base, required information is output to the user terminal, recursive return is organized if a negative result is obtained, computation results are compared to permissible values, and control is transferred to the control module.

The control module sends to the device selected by the user (terminal, plotter, display, printer) the results of computation and transfers control to the operating system monitor.

Machine time on an SM-4 computer is 5 minutes for computation of circuit board parameters, 1-2 minutes for computation of reflection noise and the same amount of time for estimation of cross talk noise.

The experience gained in using the SAPR VEMS subsystem has shown its effectiveness for the design of two-sided printed circuit boards for high-speed devices and microprocessor equipment.

REFERENCES

1. Primeneniye integralnykh mikroskhem v elektronnoy vychislitelnoy apparature (Use of Integrated Microcircuits In Electronic Computer Equipment). R. V. Danilov, S. A. Eltsov, Yu. P. Ivanov, et. al. Moscow, Radio i svyaz Press, 1987, 384 pp.

2. Posa, J. "Status and Prospects for High Speed Integrated Circuit Programs." Elektronika, 1981, 54, No. 19, pp. 80-88.

3. Knyazev, A. D. Elementy teorii i praktiki obespecheniya elektromagnitnoy sovmestimosti radioelektronnykh sredstv (Elements of the Theory and Practice of Electromagnetic Compatibility of Electronic Equipment). Moscow, Radio i svyaz Press, 1984, 336 pp.

4. Sistemy avtomatizirovannogo proyektirovaniya v radioelektronike (Automated Planning Systems In Electronics). Ye. V. Avdeyev, A. T. Yeremin and I. P. Norenkov, M. I. Peskov. Moscow, Radio i svyaz Press, 1986, 368 pp.

5. Schibuya, N., Takagi, H., Ito, K. NESSY- "Noise Estimation System for Board-Level Circuit Using Electronic Circuit Simulator." Proc. VII Intern. Wroclaw Sympos. on EMC, Wroclaw, 1984, pp. 939-946.

6. Kechiyev, L. N., Smirnov, S. N., Tsirin, I. V., "The SAPR VEMS Printed Circuit Design System." Konstruirovaniye spetsialnoy elektronno-vychislitelnoy apparatury: Mezhvuz sb. nauch. tr. Ryazan, RRTI Press, 1984, pp. 56-59.

7. Kechiyev, L. N., Smirnov, S. N. "Considering Requirements of Internal EMC In Automated Planning of Printed Circuit Boards." Elektromagnitnaya sovmestimostradioelektronnykh sredstv v podvizhnykh sistemakh, Riga, IEiVT AN LatvSSR, 1985, pp. 223-224.

8. Kechiyev, L. N., Smirnov, S. N. "Printed Circuit Board Automated Planning System Considering Internal EMC Requirements." Tr. VIII mezhdunar. vrotslavskogo simpos. po EMC [Works of International EMC Symposium], Wroclaw, 1986, pp. 676-683.

9. Operatsionnaya sistema SM EVM RAFOS (The RAFOS SM Computer Operating System). L. I. Valikova, G. V. Vigdorchik, A Yu. Vorobev, A. A. Lukin. Moscow, Finansy i statistika Press, 1984, 207 pp.

10. Kechiyev, L. N., Sukharev, P. S. "Computation of Capacitance of Printed Circuit Boards." Avtomatizatsiya proyektirovaniya i upravleniye kachestvom. Moscow, MDNTP Press, 1981, pp. 21-23

11. Kechiyev, L. N., Tsirin, I. V. "Numerical Method of Calculating Parasitic Capacitance of Electronic Elements of Complex Configuration." Tr. VII mezhdunar. vrotslavskogo simpos. po EMC [Works of International EMC Symposium], Wroclaw, 1984, pp. 695–701.

12. Tsirin, I. V., "Considering Signal Distortion In Connections During Planning Of Very Fast Digital Electronic Modules." Tr. VIII mezhdunar. vrotslavskogo sympos. po EMC [Works of International EMC Symposium], Wroclaw, 1986, pp. 764–772.

13. Ruehli, A. E. "Inductance Calculation In a Complex Integrated Circuit Environment." IBM, J. Res. Develop., 1972, 16, No. 16, pp. 470–481.

14. Kechiyev, L. N., Vasileva, Ye. A., Orlovskiy, D. E. "Reflection Noise In Data Transmission Lines and Its Computation By Computer." Avtomatizatsiya proyektirovaniya i upravleniye kachestvom. Moscow, MDNTP Press, 1981, pp. 37–41.

15. Mahmoud, A. L. "Research on Static and Pulse Noise Immunity of Integrated Digital Circuits." Zurich, Swiss Federal Inst. of Tech, 1970, 212 pp.

UDC 658.52.011.56

Use of the DIAZ DBMS To Generate Text Documents In an Integrated Computer Facility CADD System

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian (manuscript received 15 Dec 86) No 3, May–Jun 1988, pp. 79–80

[Article by Yu. N. Sakhno]

[Text] **Introduction.** In the process of creating automation systems, the CADD developer must overcome a number of system problems, the successful solution of which has far-reaching effects. Here are some of them:

— Overcoming difficulties related to the significant interrelationship of planning results at various system hierarchy levels;

— development of planning methods reducing the volume of replanning required when slight changes are made to initial data;

— development of viable planning automation systems which adapt easily to changes in hardware elements, design and other system parameters;

— flexible restructuring and access to initial data for the planning process in general, as well as each individual stage.

Therefore, at present one of the important component parts of an automated planning system is its automated information system. The use of a centralized data source and unified data base management system can solve many problems related to contradictions in and lack of compatibility of data, and also significantly decrease the cost of information servicing and storage.

System design principle. An integrated automated planning system should include subsystems to automate all stages involved in planning computer equipment — circuit design and structural design, technological preparation of production facilities.

In this article we studied a subsystem for generation of text documents for circuit and structural design.

Text document planning subsystems place special emphasis on independence of software from stored data structures and output document formats. Therefore, there must be a universal data access apparatus to assure subsystem flexibility.

The IKAR automated planning system[1] contains a subsystem for output of text documents which is constructed using the DIAZ DBMS data manipulation language. The subsystem performs the following functions (cf. Figure):

— Construction of a "pattern" element list from the results of synthesis of the electrical circuit, using certificate data on available elements from a product library;

— sorting of information on the element in accordance with the requirements of a specific text document;

— printout of documents using a pattern describing the output document format.

During synthesis of an electrical circuit, each component product is assigned a unique code, which the text document formatting programs use to construct a component product frame with the aid of the DBMS in a single examination of the standard data library.

The standard data library of component products contains the characteristics of these products and is organized as a data set consisting of various segments of variable length. In order to restrict the component product library to reasonable limits, the products used in objects to be planned were organized. Products, the numbers of types of which is limited to a few hundreds, have unique codes (for example, microcircuits, transformers, etc.). Products, the number of standard types and sizes of which is in the thousands, are grouped according to the most variable parameters.

For example, the total number of resistors which can be used is

$$S=\sum_{i=1}^{m} M\cdot E\cdot T\cdot B\cdot K.$$

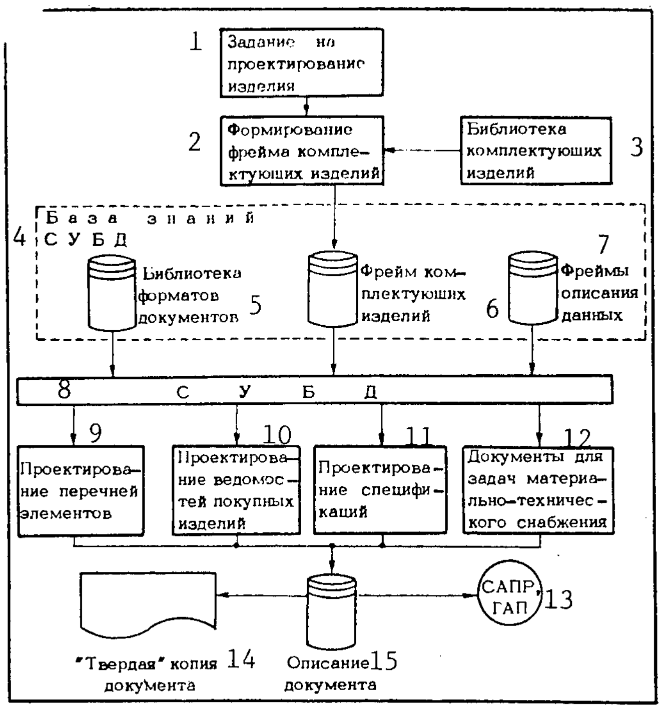
where m is the number of resistor types permitted for use (MLT VS, etc.); M is the number of standard power ratings (0.125 W, 0.5 W, ...); E is the number of gradations in the nominal units of measurement (Ohm, kOhm, MOhm); T is the number of precision characteristics for the type (0.5%, 2%, ...); B is the number of product utilization versions; K is the number of nominal ratings used.

It is obvious here that the parameters M, E, T and B can take on a few (dozens) of values, while K could take on tens of thousands of values. Therefore, grouping of products without considering nominal values decreases the number of standard data sets by several orders of magnitude. The nominal value can be used as a modifier with its value verified as to agreement with the permissible nominal values.

The use of this method for design and operation of the library of component products has significantly decreased the cost in machine time for processing of component product standard data.

The standard data for component products include the following characteristics: Full name of product, type of product, electrical parameters, foreign analogues, reliability parameters, delivery document, usage requirements, mass and dimensions.

Maintenance of the library of component products and document formats is performed by the dialog facilities of the DIAZ DBMS, allowing on-line retrieval, checking and editing of information in the database.


1
Задание на проектирование изделия
2
Формирование фрейма комплектующих изделий
Библиотека комплектующих изделий
3
База знаний
4
СУБД
Библиотека форматов документов
5
Фрейм комплектующих изделий
6
7
Фреймы описания данных
8
СУБД
9
Проектирование перечней элементов
10
Проектирование ведомостей покупных изделий
11
Проектирование спецификаций
12
Документы для задач материально-технического снабжения
САПР, ГАП
13
"Твердая" копия документа
14
Описание документа
15

Key: 1, Product planning assignment; 2, component product frame formatting; 3, component product library; 4, knowledge base DBMS; 5, document format library; 6, component product frame; 7, data description frames; 8, DBMS; 9, element list planning; 10, purchased part register planning; 11, specification planning; 12, documents for material and equipment supply problems; 13, CADD, GAP [expansion not given]; 14, hard copy of document; 15, description of document.

Document planning application programs are constructed on the basis of a specialized data access language which can be used in programs written in FORTRAN, PL or assembler[2].

When working with the data manipulation language, the user need not know the location of information in peripheral storage or its storage structure. He can access data at the level of logical segments and requisites, the names of elements in the area of interest. This allows great attention to be given to the functional specifics of programs which, in turn, generally increases the level of reliability of application programs. A tabular language is used to describe the storage structures, allowing description of a broad class of data structures. The abbreviated language syntax can be represented as Bakus-Naure records:

```
<data set>:=<set name><set type>;
<segment>:=<segment name><segment length>;
<requisite>:=<requisite name><requisite address in segment>
<logical and physical length of requisite>
<logical and physical format of requisite>
<permissible values of requisite>.
```

The physical length of a data set depends only on the type of peripheral storage medium. A set can occupy an entire disk pack. The length of a segment is determined by the size of main memory which can be used by the DBMS and may vary from 0.4 Kbytes to 10 Kbytes. The length of a requisite cannot exceed 256 bytes. The number of different segment types in a set is unlimited.

The method of conveyor processing is used to format text documents. The input of the subsystem receives a "clean pattern," containing only a description of the document format. As the pattern moves through the subsystem, each functional program, utilizing the data structure description language, inserts the needed information. To change the format of an output document, only the pattern description and data structure description language must be changed. The orientation of the program to functional data processing, independent of data structures, allows standardization of a number of programs for planning of various document types.

A significant place in the text document formation subsystems for computer equipment design is occupied by the program which sorts the names of component products in accordance with the requirements of the state standards and technical conditionis. To make the software independent of the format in which names are stored in the system, a product name structure description metalanguage has been developed. This language is used to describe the names of requisites and the sequence of their use in names. After all sorting stages are completed, the necessary requisites are combined into a single requisite.

Certain difficulties arose in the sorting of Russian-English text due to the lack of a combined table of characters of the two languages. A combined table of upper and lower case letters in the two alphabets should be created, with codes in alphabetical order. The author is in full agreement with M. I. Belyakov in this respect (cf. Appendix of[3]).

Conclusion. The organization of the system suggested in this work for operation on a type YeS1055M computer is now in use as a part of an integrated automated computer equipment planning system. Work is now in process on increasing the power of the DBMS and the text document planning system.

REFERENCES

1. Gerasimenko, Ye. P., Sakhno, Yu. N. "Parametrically Tuned Automated Planning System For Technological Masks and Topologic Drawings." Avtomatizatsiya proyektirovaniya elektronnoy apparatury, 1984, No. 3, pp. 16-19.

2. Sakhno, Yu. N., Yakovenko, L. I. "Automatic Planning System Database Used to Develop Computer Equipment for the SM and ASVT-PS Computers." Opyt razrabotki i vnedreniya tekhnicheskikh i programmnykh sredstv SM EVM i ASVT-PS (Experience In Development and Introduction of SM and ASVT-PS Computer Hardware and Software Facilities). Abstracts of Reports. Moscow, 1986, Part 2, pp. 135-136.

3. R. Gauthier Rukovodstvo po operatsionnoy sisteme UNIX (Guide to the UNIX Operating System). Moscow, Finansy i statistika Press, 1985, 232 pp.

UDC 681.3.06:681.142.2

Formula Description of Typical Topologic Elements of Microwave-Band Integrated Circuits and Interpreter Specifics

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian (manuscript received 23 Apr 86; after revision 25 Dec 86) No 3, May-Jun 1988, pp. 81-83

[Article by O. I. Amvrosova]

[Text] **Introduction.** In planning the topologies for hybrid integrated microwave-band circuits, topologies for electronic-optical system control grids in electronic vacuum microwave devices, topologies for special measurement device sensors, etc., the set of elements in the 15-UT-4-017 "Kulon" interactive graphic system is insufficient. A set of programs[1] significantly expands the capabilities of the interactive graphic system and the class of topologies planned. However, in many cases, manual formation of topologies (description of the topologies as a set of broken arcs) is quite cumbersome and requires a great deal of time. The example presented below illustrates that description of topologies using the "Kulon" standard is impossible, while description using the expanded set[1] is too cumbersome for practical application.

Furthermore, in the development of topology parameters it is frequently necessary to modify topologic elements in ways which are not supported by standard editing facilities (changing the number of turns in spiral elements, individual element parameters, the number of cross pieces in a control grid, etc.). In practice, such elements must be generated as new elements, which also significantly increases planning time.

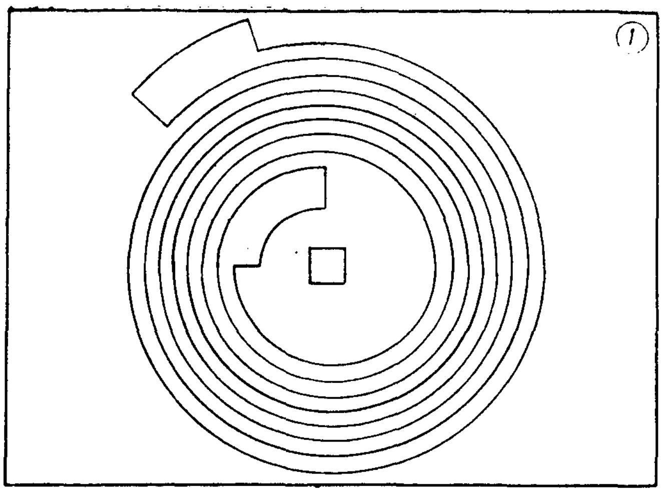
The approach suggested in this work and the further development of the "Kulon" interactive graphic system software allow generation of topologic elements based on a formula description of their geometry. The elements are described by a certain set of parameters defined by the developer, changing of which in dialogue mode produces the necessary topology modification.

Formation of topologies using parametric elements. Let us describe the facilities of the new method for generation of topologies. Let us take for example the task of generating preparation programs on the KPA-1200 device for eddy-current sensor masks for testing quality of wire and paths. This type of topology is shown in Figure 1. It was described using an approximate representation of a spiral as a curve consisting of circular arcs. To encompass the variety of such topologies and provide sufficient accuracy, four standard elements had to be introduced — spirals with different combinations of characteristics: Right hand and left hand spirals, the last incomplete turn larger than 180°,smaller than 180°.

To assign a parametric element (by which we refer to typical elements, emphasizing their dependence on parameters), the grammatical rules must be used to formulate text containing the formula description. We present below a description of one of the four elements mentioned above (shown in Figure 2). The line number is not included in the text of a description. It is used for reference in the following commentaries.

```
1. ANGLE a
2. ANGLE b
3. RADIUS p
4. RADIUS r
5. RADIUS t
6. d
7. NUMBER n
8. ANGLE TO TURN h
9. ORIGIN O
10.
11. (2 — cosa+sina — cosb+sinb)*d/8=c
12. (1 — cosa+sina)*d/8=e
13.
14.
15. P(t*cosb, t*sinb)
16. P((r+c+d*(n — 1))*cosb, (r+c+d*(n — 1))*sinb)
17. >P(—d/8, d/8)
18. n — 1
19. P(r+e+d*(n — k), 0)
20. >P(d/8, d/8)
21. P(0, —r — e — d*(n — k)+d/4)
22. >P(d/8, —d/8)
23. P(—r — e — d *(n — k)+d/2, 0)
24. >P (—d/8, —d/8)
25. P(0, r+e+d * (n — k)— 3* d/4)
26. >P(—d/8, d/8).
27. P(r + e, 0)
28. >P(d/8, d/8)
29. P(r * cosa, —r * sina)
30. P(p * cosa, —p * sina)
```

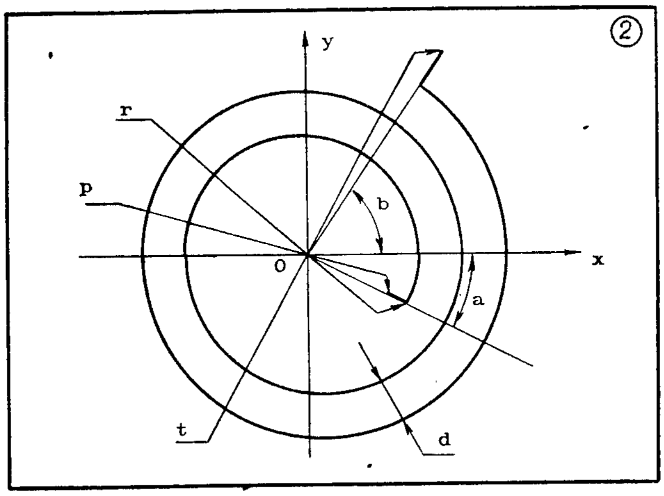
Lines 1-9 list the parameters of the first type. To generate specific topologies, these parameters are set by the user. They may be numbers (in which case their values are input through the character display) or points, i.e., pairs of numbers (in which case the parameters



Example of Eddy-Current Sensor

may be input through a character display or by pointing at the spot using a marker on a graphic display screen). Number parameters and point parameters are represented by lower-case and upper-case Latin letters. All parameters listed in these lines (except *n* and *h*) are shown in Figure 2; *n* is the number of turns of the spiral, including the last, incomplete turn; *h* is the angle of rotation of the local system of coordinates relative to the coordinate system of the drawing. The permissible values of parameters *a* and *b* are: $0<a\leqslant 90^\circ$, $0<b\leqslant 90^\circ$.

Lines 11 and 12 continue the list and define the supplementary parameters used to simplify and abbreviate the text.



Topology Fragment

The description of a parametric element is a description of its boundary as a sequence of line sections and arcs. Line pairs 15, 16 and 29, 30 describe sectors connecting points. Lines 16, 17 and 19, 19–21, 21–23, etc. describe arcs which have an origin, center and end at the corresponding points (moving clockwise). Parametric elements can be described by organizing a loop: Lines 19–26 are repeated *n*-1 times, *k* taking on values of 1, 2, ..., *n*-1.

The description lines contain formulas expressing the variation of point coordinates on parameters with respect to the local coordinate system.

The text describing a parametric element in the graphic language is the program for generation of a specific element of this type based on the assigned parameter values. Within the framework of the expansion of "Kulon," implementing the method suggested, this is done by means of a graphic editor.

The new function "parametric element" is added to the DRAW mode, to be accessed using the name of the element and values of the parameters of the element input by the user, generating the element which is then placed in the output file as a set of "line segment" and "arc" figures and is displayed on the graphic display screen.

The mask shown in Figure 1 can be described by three types of elements: A rectangle and two types of spirals.

Inclusion of additional programming facilities in the "Kulon" interactive graphic system software. The procedure for making changes to the graphic editor is worthy of attention.

Since any access to a DRAW function is preceded by a search for the code of the function in a table, adding a new function requires that its code be included together with a reference to the corresponding overlay in the table. The "arc" function was excluded from the table, which caused no loss, since an arc can be defined as a parametric element.

All remaining changes related to entry of a new function were made in the ENIC overlay. This overlay was suggested to process the new function because it processes the "cell" function and contains many subroutines which can be used to process the "parametric element."

As the modules with the ENIC overlay were configured, its text was not used. All modules included in the overlay were added as machine-language programs at the end of the overlay. When it was necessary to make changes within the overlay, the corresponding group of instructions was replaced with a new group occupying the same memory area with transfers, if necessary, to instructions placed at the end of the overlay.

When control is transferred to an overlay, redistribution by functions according to the table is performed. The code for the function "parametric element" was included (added) to the overlay function table, together with the address where the corresponding subroutine was located.

The "parametric element" function is processed by universal overlay subroutines, as well as subroutines in the resident portion of DRAW. Some of the former subroutines were modified to allow their use with parametric elements. These changes created additional capabilities for processing of other ENIC overlay functions as well. In particular, it became impossible to use two methods to assign coordinate points (see above).

Processing of a new function utilizes standard buffers and working locations, particularly the topology description buffer (where information on figures is initially generated), buffers to read and write portions of a graphic file. When working with them, the possibility must be considered of their utilization by subroutines and functions which may be accessed as the parametric element is generated and also, when possible, "damage" to them should be avoided, if necessary by saving it and restoring their contents.

Interpretation of formulas descriptions of typical graphic elements. The software implementing this function is essentially an interpreter for the formula descriptions of standard graphic elements. Let us say a few words concerning the peculiarities of the interpreter used.

Text is processed in two stages. The first stage forms special programs or preliminary processing of the text file TEST. Grammatical analysis is not performed here, rather merely the fact of inclusion of the chain of characters in the language is verified. The algorithm used is based on the principle of the finite automatan[2]. The program diagnoses the grammatical errors found: It indicates the text location where an error occurred. If an error is detected, the user must edit the text and access the program again.

The second stage occurs during processing of the new "parametric element" function, included in the DRAW mode. It includes the actual grammatical analysis and performance of actions called for in the formula description. The program uses the formulas to calculate the values of coordinates of characteristic points on a parametric element and generates the output graphic file. It is assumed in this stage that the sequence of symbols is clearly represented in the grammar, avoiding checking which would be necessary to avoid loops or accessing forbidden addresses, and also to generate and output diagnostic messages on errors if the input sequence of symbols were assumed arbitrary. This reduces program size and running time.

Conclusion. The entry of new standard graphic elements by formula description expands the capabilities of the 15-UT-4-017 "Kulon" interactive graphic system. Their use in planning the topology of microwave-band hybrid integrated circuits and in many other cases significantly facilitates the process of manufacture of masks. We should also note that this approach can be used to create systems for automated design of products based on preliminary parametric descriptions of standard structural elements, which are subsequently utilized to generate drawings of a product and its individual units.

REFERENCES

1. "Expanded Graphic Program System for the 15-UT-4-017 'Kulon' Interactive Graphic System." I. P. Pushkov, L. G. Morina, N. A. Kochetova, et. al. Elektron tekhnika, Series 1, Elektronika SVCh, 1985, No. 7, 68 pp.

2. Louis, F., Rosenkrantz, D., Stearns, R. Teoreticheskiye osnovy proyektirovaniya kompilyatorov (Theoretical Principles of Compiler Planning). Moscow, Mir Press, 1979, 656 pp.

UDC 681.324:629.78

Creation of Modern Computer Systems for Complex Ecological Studies

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian (manuscript received 01 Oct 87) 1988, No 3, May-Jun pp. 83-85

[Article by V. I. Dianov]

{txt}[Text] We shall see a basically new approach to the solution of problems related to calculation and prediction of situations in various ecological systems, as well as problems of evaluating the situation in dynamic natural environments considering technogenic actions.

Statement of problems. We need to develop an effective computing system which will satisfy the following basic requirements:

— Allow description of a regional task as a sequence of initial factographic maps, usually read by specialists in the subject area;

— automatically input maps with arbitrary contour configurations for subsequent semiautomatic classification of the maps, and also support the process of combining these maps with air and space survey data;

— analyze regions with practically any degree of detail, as dictated by the detail level of available maps relating to each factor;

— allow timely access to any point in the region in question with selection of the entire vector of factors required to perform a task;

— support automatic display of initial, intermediate and resultant data in the form of maps on color raster displays considering the requirements for cartographic data representation;

— correct data in any stage of processing using facilities for graphic dialog with a cartographic database;

— connect application tasks to the system and allow operational interaction with databases of many dimensions;

— generate report documentation as hard copies of the resultant maps on photographic film or paper.

Expansion of each of these tasks is a rather complex scientific and technical problem; therefore, let us discuss just a few key aspects, the solution of which, taken together, would allow the creation of an effective computer system and performance of a number of practically important prognostic calculations on the system.

Selection of basic computing facilities. The major starting point for development of a system of this class is selection of the computer equipment and its software.

Considering the requirements outlined above, as well as the experience which has been gained in the development of effective systems for processing multidimensional data, the bases selected was the highly productive DELTA system developed at the special design bureau for mathematical machines and systems, Institute of Cybernetics imeni V. M. Glushkov, Ukrainian Academy of Sciences. A brief description of the hardware and system software of this computer system was presented in[1], while[2] describes data processing systems based on the DELTA system.

The following advantages of the DELTA system are particularly valuable for the area in question:

— Large volumes of magnetic disk peripheral storage (hundreds to thousands of megabytes) and an effective file management system;

— rapid processing by the central processor (1–3 million operations per second);

— availability of four color raster display processors and effective raster machine graphics software;

— the capability of rapid and automatic input of images from photographic film with subsequent interactive image improvement;

— well developed digital image processing software;

— access to all system resources by a large number of users (up to 16);

— system reliability and compactness.

Geoinformation system. The methodologic principle of the method is representation of all factors related to the ecological problem as thematic maps (for example landscape, weather, vegetation and other maps). Each thematic map is correlated to the topographic map of the region in the corresponding scale. When ecological calculations are performed, the topographic map is generally not directly used, but rather serves as the source of the most characteristic terrain elements. The system therefore stores contour maps made from the topographic maps, which are always automatically superimposed with the thematic maps as they are displayed on the screen.

The thematic maps of a region consist of individual fragments (plots), usually tied in with the corresponding topographic map plot. Each fragment logically represents a rectangular scan field of points consisting of N_y lines of N_x elements each. A point of a thematic map has a description, a code in the corresponding thematic classifier, one byte in size. For example, characteristic measurement for a fragment of a mathematical map in 1:100,000 scale are N_x=N_y=700, yielding a file size of 0.49 Mbytes.

Each detailed map for a large territory contains large volumes of information, and if we consider the number of thematic maps, the total volume of data exceeds the available disk memory on the system. Therefore, the DELTA system uses automatic compression of maps before they are entered into the database, and each program reads the maps it requires using a standard system-wide raster image restoration function before the image is placed in computer memory.

The experience of industrial use of the system indicates that the compression factor of data in the data base is 10–50. Construction of a full raster map from a compressed representation occurs in real time within 1–5 seconds, which is quite satisfactory for this class of problems.

To support automatic joining of separate map plots to select data from an arbitrary territory, each plot, after entry, undergoes special geometric correction. As a result, the plot is represented as a rectangle. The system retains information concerning the linear and angular distortions within each elementary rectangle of the raster image, allowing subsequent calculations to be performed without introducing errors as a result of the new map projection.

Method for creating thematic maps. Before entry into the system, a map is classified as to its theme in accordance with the related problems. The result is a thematic map tied in with the topographic map of the appropriate scale.

The following sequence of actions is performed for each plot to tie it in with the geoinformation system.

1. Contours delimiting areas with identical classification codes are drawn onto transparent plastic film, then the copy is photographed in reduced scale. Reference points are entered on the copy for subsequent geometric correction of the map.

2. The UVVI-200 device, manufactured by Karl Zeiss Jena (East Germany) is used to generate a raster half tone map image from the photographic film.

3. The image is examined on the screen in full scale (maps are usually encoded at a resolution of 1024×1024 points), then improved by digital filtering, brightness adjusted, etc. The result is an image with contrasting figures on a black field.

4. The PAINT graphic editor is then used to color all areas in semiautomatic mode: The system selects the next "uncolored" area and requests its classification code; after the code is input by the operator it is extended to the entire area. After this, the system guides the operator to the next area, continuing until the entire plot is filled. The mean time required to color a rather well saturated 1:100,000 scale landscape plot is 45 minutes.

5. The separating contours between the areas are then erased by entering an instruction in the PAINT editor program.

6. The cursor (using the PAINT program) is then used to mark reference points, after which automatic geometric map correction is performed. As a result the plot is tied in to the standard scale ($N_y \times N_x$) and entered into the geoinformation system in compressed form.

Automatic generation of new thematic maps. The system can generate maps reflecting environmental properties of interest to specialists in graphic form. This can be performed, naturally, only if the influence of initial factors on the resulting function has been formalized. A typical example is the process of generating maps illustrating the risk of contamination of various media with various substances. In such cases, the initial information is found on maps of constant environmental factors (landscapes, vegetation, ground water depths, weather data, etc.), to which the system provides access by an application program. The program constructs values of the output function at each point, thus automatically constructing necessary thematic map in the database, which, in turn, may serve as a basis for subsequence computations and expert estimates.

Processing of half-tone images. The system allows half-tone images (such as aerial photographs) to be combined with thematic maps. The procedure for preparing a half-tone image for this application involves using the graphic editor as follows. First the image is entered by the UUVI-200 device, then brightness adaptation is performed and the required number of brightness bands is distinguished. After this, geometric correction occurs with correlation to the corresponding topographic map plot. Subsequently, there is thematic maps can be entered on the image in the combinations required by the operator. Most frequently used for these purposes are general purpose thematic maps: The river network, administrative subdivisions, highways, etc.

REFERENCES

1. Dianov, M. I. "The DELTA High-Speed Minicomputer." USiM, 1986, No. 6, pp. 6-9.

2. Dianov, V. I. "Principles of Organization of Highly Productive Systems for Processing Continuous Data Streams." Ibid, pp. 3-6.

UDC 681.3.06:621.391+681.325.657

Implementing Digital Signal Processing Algorithms On SM Computer With Specialized Fourier Processor

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian (manuscript received 18 Aug 86) No 3, May-Jun 1988, pp. 86-88

[Article by Yu. N. Aleksandrov, B. M. Katkov, V. V. Kolobkov and A. V. Lazarev]

[Text] **Introduction.** Digital signal processing methods are widely used in many areas of science and technology. With massive automation, practical implementation of digital processing became possible with the appearance of highly productive, inexpensive computer hardware, as well as specialized devices for data processing with very high speed and accuracy.

Implementation of spectral-correlation analysis and filtration algorithms occupies a special position in digital processing[1]. The area of their application is quite broad — from radar to medical diagnosis[2]. However, implementation of algorithms of these types requires large numbers of computations, since Fourier transforms must be computed. This places limitations on the use of universal mini- and microcomputers for real-time digital signal processing and requires the use of specialized devices to speed up the execution of the Fourier transforms so-called Fourier processors. The throughput of such devices may exceed the capability of universal computers for Fourier analysis tasks by two or three orders of magnitude.

Both in our country and abroad, intensive work is under way on the creation of various Fourier processors, and, in particular, processors for real-time signal processing. Development of a radar Fourier processor has been announced, implementing a fixed algorithm on samples read at a frequency of 30 kHz. Universal Fourier processors with throughput capacities of several kilohertz to 1 MHz are also manufactured[1].

A broad class of experiments can be distinguished in scientific research, requiring investigation of electric signals in the 1-20 kHz frequency band (for example, in geophysics and seismology, radio astronomy and radar, as well as analysis of mechanical vibrations[2], speech signals[1], cardiograms and encephalograms in medicine). These experiments require real-time (i.e., as the signal arrive) recording and processing of information, the volume of which exceeds the memory capacity of modern small computers.

The SM-1300.1705 computer system with Fourier processor, manufactured at "Energopribor" experimental plant, is intended for real-time digital signal processing. This article describes the software created for the system by the authors.

Basic characteristics of digital system with Fourier processor. The digital computer system is designed for a broad range of problems related to signal investigation. The structure of the system is shown in Figure 1.

An important element of the digital system with Fourier processor is its highly productive specialized Fourier processor. This device rapidly performs various versions of spectral analysis, calculates mutual spectra, performs nonrecursive filtration and certain supplementary functions.

The specialized Fourier processor contains a direct access interface which supports communication between the specialized Fourier processor control processor and the common bus of the SM computer. The specialized Fourier processor was described in detail in[3-5].

The system includes a combined high speed device (UKB-200). This device can digitize signals at a rate of 1 MHz. The UKB-200 controller includes a direct access channel supporting interaction between the voltage–code converter and the SM computer common bus.

Digital system with Fourier processor software structure. In developing the software, the following requirements were considered:

— Support for processing and recording of results at the rate of input signal arrival, as well as simplicity of user of the digital system with Fourier processor;

— development of software facilities for effective utilization of the specialized Fourier processor and the UKB-200;

— implementation of software as an application program package to run under the RAFOS operating system.

The structure of the software is shown in Figure 2. The program package is written in assembler and FORTRAN. Work with the software is organized in dialogue mode, allowing variation of parameters (digitization frequency, transform size, analog information input channel), as well as interruption and restarting of processing as desired.

To decrease the time required to perform processing and the complexity of distribution of memory, interaction among the UKB-200, specialized Fourier processor and SM computer memory is organized at the physical level.

The sequence of operation of the digital system with Fourier processor programs is as follows: Analog signals enter through the analog data input channels of the UKB-200 to the voltage–to–code converter unit (cf. Figure 1) and are converted at the assigned digitization frequency to sequences of digital codes, which are sent from the voltage–to–code converter data register by the controller through the direct memory access channel into RAM. From computer memory, the information is sent through the SPD? by the control processor and then returned to RAM, from which, after coding, is stored on magnetic tape.

The software package is a set of modules (RAFOS system and application program modules) supporting the beginning, maintenance and termination of exchange among the SM-1300.01, specialized Fourier processor and UKB-200.

Writing the software package. The basic functions of the software package includes:

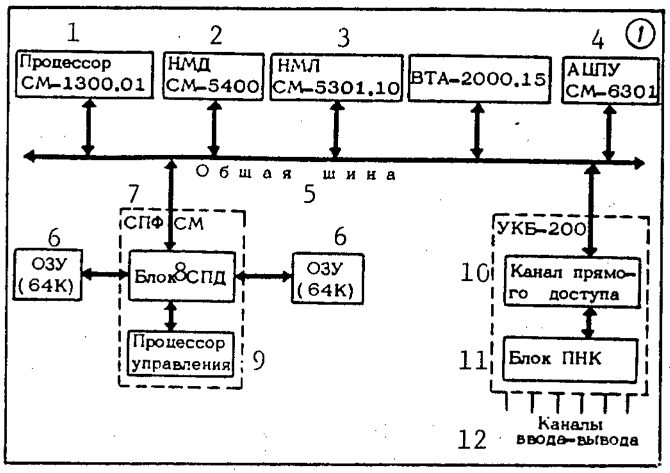
— Assignment of type and mode of processing (loading of direct access interface address registers containing the information required to organize exchange of data between SM-1300.01 RAM and the specialized Fourier processor, specialized Fourier processor registers through the control processor and loading of the UKB-200 address registers, setting the necessary digitization frequency and length of the file to be processed);

— information input and output (input to memory and buffering of data files after digitization in the voltage-to-code converter, storage of control information in the instruction and direct access interface status registers and starting of the exchange and processing, recording of the results of processing on magnetic tape, processing the end of transfer);

— performance of transforms (computation of direct and inverse digital Fourier transform, weighting of arrays, scaling of data, computation of the products of arrays, computation of spectral powers, as well as computation of the correlation function convolution).

The system device used is a disk, the medium on which the results of processing are stored is magnetic tape.

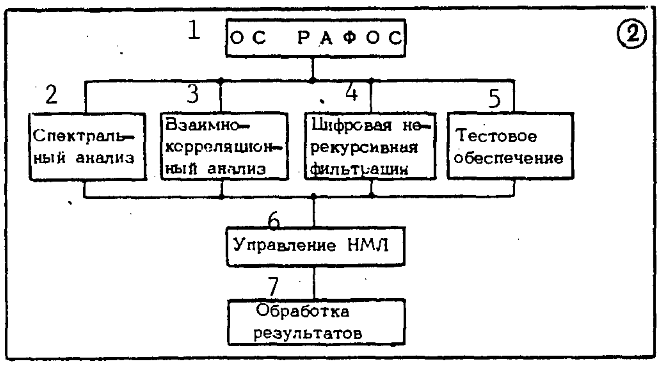
Spectral analysis is used to computer the power spectrum of the signal studied. The algorithm is based on the digital Fourier transform, implemented in hardware as a fast Fourier transform (FFT) algorithm in the base 2[1]. The power spectrum is computed with accumulation. The program supports processing of an actual signal in real time (i.e., the processes of data input, digital Fourier transform computation and results recording occur in parallel, without loss of input information).



Structure of Digital System With Fourier Processor Circuit. 1, Processor; 2, disk drive; 3, tape drive; 4, printer; 5, common bus; 6, RAM; 7, specialized Fourier processor; 8, direct access interface; 9, control processor; 10, direct access channel; 11, voltage-to-code converter; 12, input-output channels.

The mutual correlation analysis program computes the correlation function of two analog signals. The algorithm of this program utilizes the digital Fourier transform and the Plancherel convolution theorem[2]. The direct and inverse digital Fourier transform operations, as well as the joint product of two arrays are computed in hardware in the specialized Fourier processor. The program allows real-time processing.

The digital nonrecursive filtration program is designed to compute the convolution of the initial signal and a filter. This convolution is computed similarly to the mutual correlation. The digital Fourier transform, inverse digital Fourier transform and multiplication of two arrays are performed in hardware by the specialized Fourier processor.



Structure of Software. 1, RAFOS operating system; 2, spectral analysis; 3, mutual correlation analysis; 4, digital nonrecursive filtration; 5, test support; 6, tape control; 7, results processing.

The programs for operation with the tape drives control magnetic tape drives and support output of processed information to a printer or video terminal to allow analysis of results produced. Programs in the tape drive subsystem allow the user to enter information in dialog mode: Rewind magnetic tape, read, skip several blocks and return several blocks.

The data processing subsystem includes modules which support reading and output of processed information to terminal or printer for each program.

The programs in the package require no special preparation before use, allow the use of additional programs for operations with the magnetic tape and for display of information on graphic displays or plotters. Dialog mode allows the user to set the required processing mode and trace the processing as it occurs.

Three test problems are used to verify the correctness of operation of each program in the package. The characteristics of the package are presented in the table.

Digital System With Fourier Processor Software Package Characteristics

Programs	Output Parameters	Throughput Capacity
Spectral analysis	Frequency resolution (up to 1/4096)	10 kHz (combined readings)
	Real/complex signal	20 kHz (real readings)
	Signal type: Sine wave/noise Averaging of results (4)	
Mutual correlation analysis	Delay (up to 1024) Averaging (up to 4)	Two signals of 5 kHz each (real reading)
	Signal type: Sine wave/noise Correlating signals - real	
Nonrecursive filtration	Frequency characteristics (up to 1024)	5 kHz (complex readings)
	Real/complex signal	10 kHz (real readings)
	Output readings (up to 2048) Thinning after filtering (1 in 4)	

Note: Quantity of information recorded limited by RAM capacity. Output parameters identical: Size of array stored 1/4 input array size, recorded on magnetic tape.

Conclusion. The experience gained in creating this system leads to the following conclusions.

1. The current state of development of microprocessor engineering and specialized Fourier processors and the construction on their bases of digital system with Fourier processors have resulted in additional advantages over the use of universal computers of comparable throughput capacity:

— Decreased system cost with superior characteristics;

— increased system reliability, decreased requirements on operating conditions, reduced dimensions and simplified servicing;

— reduced data input-output time and simplified processing algorithm.

2. The characteristics of the digital system with Fourier processor software package indicates that this system could be widely used for real-time digital signal processing in various areas of science and technology.

REFERENCES

1. Rabiner, L., Gould, B. Teoriya i primeneniye tsifrovoy obrabotki signalov (Theory and Application of Digital Signal Processing). Moscow, Mir Press, 1978, 848 pp.

2. Maks, Zh. Metody i tekhnika obrabotki signalov pri fizicheskikh izmereniyakh (Signal Processing Methods and Techniques for Physical Measurements). Moscow, Mir Press, 1983, 568 pp.

3. "Specialized Processor for Fast Fourier Transforms and Signal Processing." B. Ya. Feldman, I. A. Kopyto, A. Ya. Pesin and A. M. Reynberg. Opyt primeneniya v narodnom khozyaystve mini- i mikro-EVM semeystva SM EVM (Experience of Use of the SM Series of Mini- and Microcomputers In the Economy). Abstracts of Reports, Moscow, TsNIITEIP Press, 1984, pp. 19-20.

4. Feldman, B. Ya., Krylov, G. A. "A Specialized Processor for Fast Fourier Transforms and Signal Processing." Tr. INEUM Press, 1983, No. 100, pp. 104-108.

5. "A Specialized Fourier Processor and Its Basic Capabilities." B. Ya. Feldman, I. A. Kopyto, A. Z. Krushinin and A. M. Reynberg. Ibid, 1984, No. 103, pp. 8-11.

UDC 681.3.06

Application Program Packages of the "Neyron I9.66" Personal Computer

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian (manuscript received 28 Sep 87) No 3, May–Jun 1988, pp 89–91

[Article by N. P. Starodub, A. I. Slobodyanyuk, S. D. Pogorelyy, S. G. Vaysband]

[Text] The effectiveness of personal computers currently manufactured by Soviet industry is largely determined by the level of development of their application software. A number of application program packages have been developed and are effectively used by personal computers based on the 16-bit LSI series K1810 chips ("Neyron I9.66," YeS1840, "Iskra-1030"). The compatibility of these computers with each other and with the *IBM PC* (USA) allows them to use software packages developed both here and abroad.

Application packages are a class of software oriented toward the user. They usually function under the control of an operating system (OS), using all of its capabilities to organize input–output operations and interactions with PC hardware resources. The user need not be familiar with the details of the organization and structure of the PC and OS. These packages take upon themselves the functions of organizing interaction with the user. This allows a broad range of nonprofessional users to take advantage of PCs.

Depending on the area of application, application programs are divided into two classes — general purpose and task-oriented.

The first class includes programs designed to perform tasks which arise in many spheres of human activity (creation and management of text documents, automation of computations, management of information files, processing of graphic information, etc.). The second class includes application packages designed to perform special computing tasks (automation of printed-circuit planning, automation of processes related to technological preparation of production, planning of programs for numerically controlled machine tools, operations in technological networks).

We describe below a few application program packages developed for the "Neyron I9.66" personal computer[1]. As we have noted, they can function without modification on other personal computers in this class as well.

Let us look at some general purpose programs[2, 3].

NEYRON-TEXT is designed for the creation, editing, formatting and printing of a wide variety of text documents — books, reports, explanatory notes and technical documentation.

With the advent of PCs, the use of "menus" expanded. A menu is a list of options divided by the program for the user, which is displayed on the screen. Selection from a menu is usually performed by pressing a single key. Menus significantly facilitate the use of computers by nonprofessional users and create maximum convenience. NEY-RON-TEXT makes extensive use of menus, not only at the main (top) level, appearing when the program is started, but also in each mode (editing, formatting, printing).

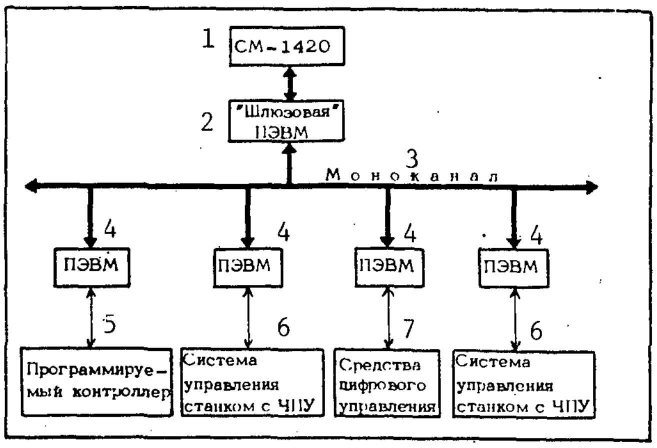
Another advantage of NEYRON-TEXT is its active use of the function keys (F1-F10), programmed in this package to perform a wide variety of editing functions (insertion and deletion of lines and blocks of text, search for and replacement of a line of text entered, storage of a block of text in a requested file or reading of text from a file, etc.). Due to the broad capabilities for formatting text (determining page length, line length, justification of the right margin, alteration of spacing between lines, printing of portions of the text in bold type, etc.), the quality of documents produced can approximate printed text.

The experience gained in using NEYRON-TEXT has shown its convenience and effectiveness for generation of large volumes of documents. This program was used to develop the documentation for the "Neyron I9.66" computer. Future development of the package will be directed toward inclusion of facilities for creation of design documentation in accordance with the unified standard for design documentation.

NEYRON-SCHET [CALC] provides the user with simple and convenient means to perform a wide variety of computations in table form. From the mathematical standpoint, the program displays F in the form

$$F=M\rightarrow N,$$

where M is a matrix of formulas, each position of which contains a computing algorithm; N is the set of real numbers.


1 СМ-1420
2 "Шлюзовая" ПЭВМ
3 Моноканал
4 ПЭВМ
4 ПЭВМ
4 ПЭВМ
4 ПЭВМ
5 Программируемый контроллер
6 Система управления станком с ЧПУ
7 Средства цифрового управления
6 Система управления станком с ЧПУ

Key: 1, SM-1420; 2, "gateway" PC; 3, single channel; 4, PC; 5, programmable controller; 6, machine tool controller; 7, digital controller.

The user can enter into any position in the table a label, a number or a formula, fixing the relationship between other positions (those located above and to the left of the position containing the formula). In addition to formulas entered by the user, a library of functions built into the program, including a number of arithmetic, logic and trigonometric functions, may be used. The library is open for addition. The package features convenient methods for movement around the table, insertion and removal of rows and columns, changing output format (numbers can be displayed with any desired number of decimal points, in exponential form, or in diagrammatic form). It is possible to repeat formulas in various parts of the table with identical sends, automatically adjusting the table positions to which they refer. The display screen can be divided into two windows, showing fragments of two different tables. This program can be widely used for engineering and scientific and technical calculations, as well as processing of economic and bookkeeping information.

NEYRON-BAZA is a relational database manager. Connections between data elements in the relational database are organized as two-dimensional labels. The database structure is defined by the software package in interactive mode. The user sets the names of fields, their types (character, numeric or logical) and length.

After the structure is defined, data are entered into the database. The package contains facilities for editing, insertion and deletion of records, as well as modification of database structures. Different databases can be copied or merged. When working with databases, it is frequently necessary to sort them according to some key (field).

NEYRON-BAZA allows sorting by both numeric and character fields.

Information which has been processed can be printed out as documents in desired formats. The format is also determined in interactive mode (header, column headings, their width, information content).

The package can be used to create and manage information-reference and information-retrieval systems, local databases of various types (frame, technological, bookkeeping).

NEYRON-FAYL is designed to manage information in the form of records. The package functions in menu mode and is basically designed to automate management activities, print correspondence, output documents of identical format using information stored in a file (notices, service records, agendas, etc.).

Records can be sorted according to any field, subsets can be defined satisfying a given condition, documents can be printed in desired formats. The package can be used in library and patent work, to automate personnel documentation, account for materials and products, etc.

Large numbers of task-oriented program packages are now under development, intended for the solution of special data processing problems. We shall now discuss one of them, supplied with the "Neyron I9.66" computer.

NEYRON-KOP is designed to control instruments and equipment in automated measurement systems using a general-purpose input-output channel as specified by state standard GOST 26.003-80. NEYRON-KOP is based on a BASIC language interpreter.

The package includes a BASIC interpreter and a set of special subroutines written in BASIC designed to organize exchange of information and control the general purpose channel interface and to test the interface.

One of the most important features of the BASIC interpreter from the standpoint of exchange and control of the channel is its well developed string-handling facilities, allowing isolation of the required portion of a string, comparison of strings, merging of strings, conversion of a number to a string and vice versa.

These capabilities are used to select and process the required data in the sequence of messages sent through the channel. Another important feature of the BASIC interpreter is the presence of graphic operators and color control operators. This allows information received over the channel to be displayed in graphic form and in color.

Furthermore, the BASIC provides the user with well developed facilities for use of program files, as well as direct and sequential access data files. The language includes facilities for direct access to input-output ports and memory locations.

This entire variety of powerful language facilities allows effective utilization for control of instruments in an automated measurement system.

Let us list the major functions of NEYRON-KOP:

— Initiation of the general purpose channel (this function must be performed before beginning operation with the channel);

— transmission of information over the channel to a device with a desired number (device number, data transmitted and delimiter code are transmitted as named variables);

— transmission of information through the channel in nonaddressed mode;

— input of information through the channel from a device with a desired number;

— input of information through the channel in nonaddressed mode;

— setting and clearing of a remote control signal;

— querying a numbered device (the result is the device status byte);

— transmission of a single byte instruction through the channel.

NEYRON-KOP provides the user with the ability to check the operation of the channel interface. For this purpose, the package includes a special test program. If errors are found in the process of testing the channel interface, an error message is sent to the screen.

All of these application program packages are included in NEYRON-DOS1 and operate under its control. However, they can also operate under the control of other operating systems in this class.

Many second generation application packages designed for use in local personal computer networks are now under development.

The general technology of such networks is shown in the Figure. The top level machine is an SM-1420-class machine, which, using a "gateway" PC, is connected to the local PC network, using a single channel as the data transmission medium. The PC local-area network implements the second level of computing devices, the main function of which is planning of facilities for controlling equipment at the bottom level. The bottom level includes programmable controllers, numerically controlled machine tools and other direct digital control devices.

The variety of second level PC software packages is determined by the type of computer used.

REFERENCES

1. "The 'Neyron I9.66' Personal Computer." S. D. Pogorelyy, A. I. Slobodyanyuk, A. Ye. Suvorov and A. A. Yurasov. Mikroprotsessornye sredstva i sistemy, 1986, No. 4, pp. 16-19.

2. Wesson, R. B. "Perfect Calc." - Perfect Software, Inc., 1983, 354 pp.

3. Wilson, C., Skiles, D. "Volkswriter for the CDP MPC Version 2." Lifetree Software, Inc., 1982, 92 pp.

UDC 682.3.06

Software Package for Numerical Synthesis of Dynamic Subjects In Automated Information Systems

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian (manuscript received 20 Oct 86) No 3, May-Jun 1988, pp. 91-93

[Article by S. M. Bukharova, A. F. Yeremin, Ye. K. Nikonov and A. V. Yakovlev]

[Text] One component part of any automated information system (such as a training stand or CADD system with well developed visualization system) is the subsystem supporting interaction between the human operator and the visual medium (external visual situation simulator).

The concept of an automated system with a "smart" visualization subsystem is based on the assumption that the operator, through his visual analyzer, receives the greatest volume of information, required to control the dynamic process in real time. We shall analyze a new set of software modules intended to solve the problem of visualization in a universal computer. The approach suggested can be used for digital synthesis, for example, of an image of a process of relative three-dimensional motion of several objects.

Problem description. A dot-matrix image of a dynamic space (subject) is synthesized, serving as the basis for precise generation of object representing the visual situation in a digital synthesis system. The statement of the problem can be reduced to the following: In an inertial coordinate system, lines of sight, the laws of motion of objects and certain sets of characteristic points are assigned, defining the surfaces of the initial positions of the objects in the corresponding subset. In accordance with the change in the spatial position of the lines of sight, the following tasks must be performed as a function of real (discrete) time:

— Imitate the instantaneous positions of the objects visualized in the coordinate system coupled with the lines of sight (only rotational motion of objects is simulated, the range of angular rates of rotation being $0.01-10^\circ/s$; the spatial position of the vectors of angular rotation of the objects is arbitrary);

— project a set of points defining the instantaneous positions of objects onto a mapping plane coinciding with the plane of the display screen or optical observation instrument;

— construction on the graphic display screen the corresponding object image phases of the synthesized subject, to imitate the visual space presented to the operator during training, and change it without noticeable flickering.

Problems of the development and application of synthesizing visualization systems were discussed, for example, in[1-2]. However, as a rule these works utilize the vector-matrix method of describing motion in the space of modeled objects.

However, the use of the mathematical apparatus of quaternions for this purpose[3], a more adequate mechanism for perception of the visual situation model, saves time in the execution of the required computations. This is because the line of sight to an observed object is identical with the vector portion of a quaternion — the rotation transform operator, while the angular motion of an object is interpreted as the vector-function of finite rotation.

Digital synthesis of a dynamic visual subject on a graphic display screen can be divided into the following major stages.

1. Development of the data base of modeled objects (sets of coordinates of characteristic points in the coupled coordinate system) and loading of data into computer memory. Assignment of the initial positions of objects in the inertial coordinate system.

The images of objects on the graphic display screen are generated by points. Each imaged point of each object visualized is the end of a vector $\mathbf{R}_{i,j}$ of a certain vector set, fixed a priori in the form of a data array or three-dimensional matrix, which is a dot-matrix model of the subject being synthesized.

2. Computation of the direction of a line of sight and the dimensions of the frame window. Determination of the values of control signals for the situation model (consisting of the instantaneous angular rotation rate components of the objects synthesized).

3. Generation of quaternions — rotation transform operators $\Lambda_i(t)$, fixing the positions of the coordinate systems rigidly coupled with the dynamic objects simulated in the inertial coordinate system.

Real rotation of the synthesized objects is approximated by a sequence of successive discrete rotations. The resulting three-dimensional rotation as a function of discrete time during simulation interval k is then determined by the quaternion product

$$\Lambda_i(kT) = \Lambda_{i,\nu}(kT) \circ \Lambda_{i,\nu-1}(kT) \circ \ \ldots \ \circ \Lambda_{i,1}(kT),$$

where i=1, 2, 3, ..., n is the index of the ith object in the modeled subject; ν is the number of successive rotations of the object.

The kinematic equations of rotation are assigned in Rodrig-Hamilton parameters, i.e., in quaternion form

$$\Lambda_i(kT) = \frac{1}{2} \Lambda_i(kT) \circ \omega_i(kT),$$

where ω_i is the relative angular velocity vector of object i.

4. Extraction from memory of information on the synthesized subject and computation of the new coordinates of points for the visualization object.

Conversion of the position of each imaging point is performed as the operation of quaternion multiplication of vector $\mathbf{R}_{i,j}$ to the left and right by the quaternion (direct and coupled with Λ_i) at each of the fixed moments in time of the visual situation modeling interval:

$$\mathrm{R}'_{i,j}(kT) = \Lambda_i(kT) \circ \mathrm{R}_{ij}(kT) \circ \Lambda_i^{*}(kT),$$

where $\mathbf{R}_{i,j}$ is the initial position of a certain vector from the set defining the visualization object in simulation interval k; $\mathbf{R}'_{i,j}$ is its final position in this interval; j=1, 2, 3, ..., m is the index of point j, defining the visualization object.

5. Exclusion of all point, line or plane elements of objects which are either outside the field of observation, or are hidden by other elements of the same object located closer to the observer, or by other objects closer to the observer.

6. Computation of the coordinates of two-dimensional perspective images by the method of hypercomplex mapping of points in the array of the synthesized subject on the mapping plane[3].

7. Reproduction of the signals of the resulting image, i.e., the model of the subject, on the graphic display screen as described in[4].

Purpose and usage conditions of the software package. The package which has been developed is designed to perform quaternion digital synthesis of dynamic subjects in an automated information system with sequential definition of visualization surfaces.

The package is a set of program modules organized into a single system, based on structuring of quaternion digital image synthesis algorithms. Individual modules can be used to assemble programs corresponding to various methods of performing this class of tasks.

The programs in the package can generate two-dimensional perspective images of a visual space on a graphic display screen, and can also determine the coordinates of lines of sight, quaternion-operators of rotation transforms and instantaneous axes of rotation of the objects synthesized at any moment in time in the modeling interval.

The programs of the package were written in *FORTRAN-IV, PL/1* and assembler.

The packages intended for use on the YeS computers with at least 512 Kbytes of memory under OS YeS. The package requires two YeS5061 disk drives, a type YeS7033 printer and YeS7064 graphic display, plus a YeS7052 plotter.

Structure of the package, input and output data. A general diagram of the dynamic object digital synthesis package is shown in the figure. The software modules of the package meet all the requirements for visualization system software, including modularity and expandability.

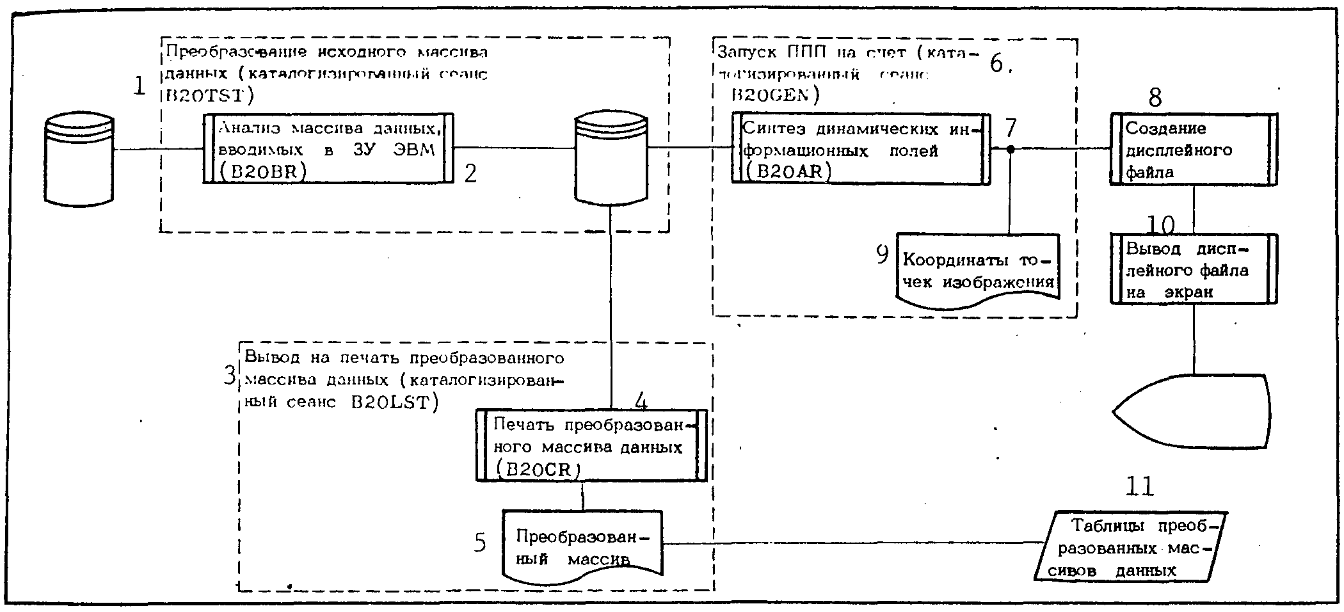
Program module B20BR, in *PL/1,* analyses the initial data arrays loaded into computer memory, counts the points of the arrays which define the initial positions of the object synthesized, and arranges the arrays in tables in A4-format pages.

Software module B30CR, in *PL/1,* counts the data in each memory page and prints the arrays of points of the synthesized subjects generated by the previous module.

Software module B20AR for digital quaternion image synthesis performs stages 3-6 in accordance with the above equations, and is written in assembler.

Creation of a display file and its output to the graphic display screen are performed by modules written in assembler.

Three catalog sessions are used in the package for digital synthesis of dynamic objects: B20TST, which converts the initial data arrays; B20LST, which outputs the transformed data arrays to the printer; B20GEN, which starts the computation package.


1
Преобразование исходного массива данных (каталогизированный сеанс B20TST)
Анализ массива данных, вводимых в ЗУ ЭВМ (B20BR)
2
Запуск ППП на счет (каталогизированный сеанс B20GEN)
6.
Синтез динамических информационных полей (B20AR)
7
8
Создание дисплейного файла
9
Координаты точек изображения
10
Вывод дисплейного файла на экран
3
Вывод на печать преобразованного массива данных (каталогизированный сеанс B20LST)
4
Печать преобразованного массива данных (B20CR)
5
Преобразованный массив
11
Таблицы преобразованных массивов данных

Key: 1, Transform of initial data array (catalog session B20TST); 2, analysis of data array input to computer memory (B20BR); 3, print out of transformed data array (catalog session B20LST); 4, print out of transformed data array (B20CR); 5, transformed array; 6, starting of software package computation (catalog session B20GEN); 7, synthesis of dynamic information fields (B20AR); 8, creation of display file; 9, coordinates of imaged points; 10, output of display file to screen; 11, tables of transformed data arrays.

The input data of the quaternion digital synthesis task include:

— The number of visualized objects;

— the common modeling time interval;

— the position of the operator–observer (direction of the line of sight) during the modeling interval — a quaternion function determining the relationship between the screen and the inertial system of coordinates, as well as the distance from the quaternion plane coinciding with the focal plane of the optical instrument used by the observer to the coordinate origin of the inertial coordinate system;

— the dimensionality of the arrays of points defining the synthesized objects of the modeled subject;

— the point arrays assigning the positions of objects before the beginning of modeling;

— other parameters included in the package but not requiring definition (if they are not assigned, standard values will be used).

The initial data on the objects synthesized have a three dimensional structure and are assigned by listing the coordinates of points on lines (sectors) or phases (polyhedrons).

As a result of solution of the problem, a two-dimensional subject image is generated on the graphic display screen. The listing generated by the plotter contains information on the positions of the instantaneous axes of rotation of the object visualized, values of components of the rotation quaternion transforms, distances from points describing the objects synthesized to the quaternion plane, as well as the coordinates of vectors defining the spatial position of objects in the subject at any moment in time during the modeled interval.

Conclusions. The software package suggested for quaternion digital image synthesis of dynamic objects requires 1.5–2 times less computer memory than the matrix method, while the computer central processor time required to process quaternion digital synthesis software modules is 40–60% less than that required for vector–matrix software modules.

The software package performs the task of digital synthesis of a dynamic visual space in simplified form, since only rotary motion of the modeled objects is represented (for example when the operator works in compensation mode).

One essential qualitative advantage of quaternion algebra as used in the software package is the ability to represent two physical quantities describing the planar rotation of an image in the quaternion plane in a single form, adequate to the spatial rotation of the imitated object: The relative angular velocity vector (local motion characteristic) and the final rotation vector (characteristic of object position).

REFERENCES

1. Smith, D. T., Matematicheskoye i tsifrovoye modelirovaniye dlya inzhenerov i issledovateley (Mathematical and Digital Modeling for Engineers and Researchers). Moscow, Mashinostroyeniye Press, 1980, 271 pp.

2. Fox, A., Pratt, M. Vychislitelnaya geometriya. Primeneniye v proyektirovanii i na proizvodstve (Computational Geometry. Applications In Planning and Production). Moscow, Mir Press, 1982, 298 pp.

3. Branets, V. N., Shmygalevskiy, I. P. Primeneniye kvaternionov v zadachakh oriyentatsii tverdogo tela (Use of Quaternions In Solid Body Orientation Problems). Moscow, Nauka Press, 1973, 317 pp.

4. Bodner, V. A., Zakirov, R. A., Smirnova, I. I. Aviatsionnye trenazhery (Aviation Trainers). Moscow, Mashinostroyeniye Press, 1978, 191 pp.

UDC 681.3.06

Software Architecture of System for Monitoring Environmental Status and Public Health

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian (manuscript received 30 Sep 86) No 3, May-Jun 1988, pp. 93-97

[Article by L. V. Kokoreva, A. N. Zhulanov, V. A. Kornelyuk and A. N. Rekhin]

[Text] Successful development of scientific and technical progress requires that its effects be monitored. This is particularly necessary in terms of the effect of scientific progress on the environment and public health in areas with intensive economic activity. The solution of this problem requires the development of a set of "industry-environment-man" models, providing the greatest possible reliability of evaluations with the minimum number of initial parameters, and also permitting prediction of the development of an entire region.

Development of such a system of models and conduct of monitoring on its basis are possible only where an automated environmental quality and public health monitoring system is used. Such a system should not only analyze the extremely complex interactions among the surrounding (industrial) environment and health within the framework of the region involved, but also must allow periodic adjustment of models to reflect the dynamics of the objects involved and the specific conditions encountered in different regions.

In the opinion of the authors of[1-3], the structure of such a system for monitoring the status of the environment and public health must include the following main segments: A database, statistical analysis software package, and a bank of models. The system should function in two modes: Experimental and production.

In experimental mode, the major modules of the system would interact as follows: A group of investigators performs statistical analysis of the information in the database to develop a model of the interaction of the environment and the health of the individual and the entire population of the region. The models, described as algorithms, are coded and placed in the bank of models.

In production mode, models would have direct access to the information in the database. The results of modeling could also be stored in the database. In production mode, the status of the environment and public health in the selected region are monitored in real time.

Some experience has now been gained in the operation of a system for monitoring the status of the environment and public health in the experimental mode in the area of preventive screening of the population. A system has been introduced and is in successful use in a number of medical-biological organizations. The system runs on the type YeS computers (under OS YeS 4.1 or later), requiring 160K of memory. The software can be used in either batch or dialogue mode.

The database. The database refers to the system of software, linguistic, organizational and hardware devices intended for a centralized storage and shared use of data.

The database in question, on massive preventive screening of the population, functions under the control of the OKA DBMS. The specific goals and requirements placed on the subject area database prevented the use of software environment of this system, which is unsuitable for the nonprogrammer-user (particularly, the research physician). A task-oriented software package for massive preventive screening of the population (called MASOB) was therefore developed, supporting the following main functions:

— Creation and management of the preventive screening database, meeting the current requirements of consistency, reduced redundancy and independence of data from application programs;

— retrieval of data based on any conditions generated by nonprogrammer-users, using a simple query language or menu system;

— generation of medical reporting documents, plus elementary statistical data processing;

— performance of research (including statistical) tasks by joint utilization of the MASOB and statistical data processing (OAD) packages.

The MASOB software package supports interaction of users, as well as the database administrator, with the database system, including the main preventive screening database and a supplementary recoded database. Both of these databases have a three-level hierarchical logic structure.

At the present time, the main database contains data on the following types of segments: Patients identification data (segment name — PAT); general information on the examination (INSP); data on patients employment (W); case history (A); social history (CA); biorhythm characteristics (BR); anthropometry (AT); biochemistry (BC); immunologic status (IM); results of examination of cardiovascular system — data from electro-, ballisto-, seismocardiograms, arterial pressure and heart rate before and after exercise, etc. (H); results of machine processing of heart rhythm (M); information on illnesses (IL).

The recoding database acts as a dictionary, supporting interpretation of coded values of parameters and definition of field format characteristics.

Initial information can be input to the database in both batch and dialogue modes. When data are input in batch mode, a specialized input form or "health status evaluation card" is used. The card is filled out by the physician during the examination, then converted to machine-readable form and input to the database.

The architecture of the MASOB software package is shown in the figure. All programs in the package are arbitrarily divided into two groups: Programs for creation and management of the database, primarily used by the database administrator, and user information service programs.

The database creation and management programs include:

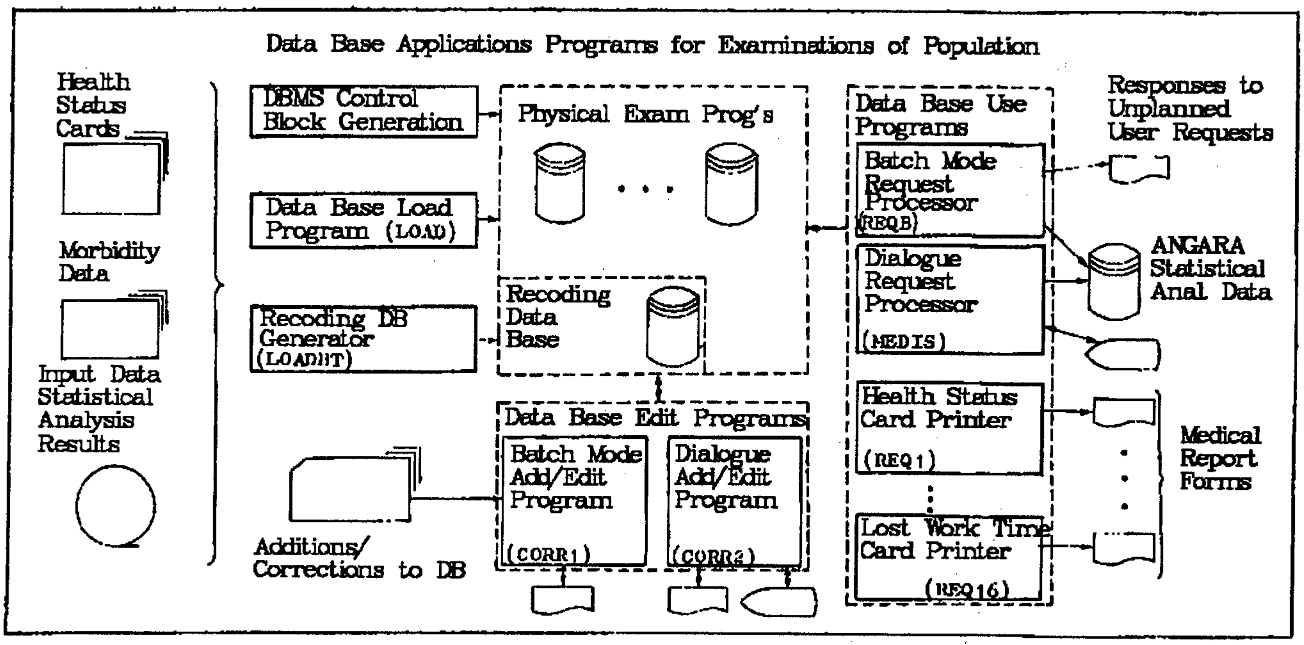
LOAD — program to load the main database from the health status cards;

LOADBT — program to load the recoding database (using special blanks);

PROSM — program to examine the loaded database. Operation of this program generates a print out of sequence of database records;

CORRTV — program to edit database records, allowing insertion, removal and modification of records in "man (database administrator) — machine" dialogue mode. Before editing the database, a search must be conducted using assigned keys. A user (the database administrator) who knows the keys can immediately find the desired record and edit it. If the keys are not precisely known, they can be refined in the database examination mode. The database administrator assigns the maximum and minimum values of the keys, thus defining a search range. Finding a record with the desired values of the keys, the administrator then switches from examination to edit mode;

CORR — program to edit database records in batch mode.


Data Base Applications Programs for Examinations of Population
Health Status Cards
Morbidity Data
Input Data Statistical Analysis Results
DBMS Control Block Generation
Data Base Load Program (LOAD)
Recoding DB Generator (LOADIT)
Physical Exam Prog's
Recoding Data Base
Data Base Edit Programs
Batch Mode Add/Edit Program (CORR1)
Dialogue Add/Edit Program (CORR2)
Additions/ Corrections to DB
Data Base Use Programs
Batch Mode Request Processor (REQB)
Dialogue Request Processor (MEDIS)
Health Status Card Printer (REQ1)
Lost Work Time Card Printer (REQ16)
Responses to Unplanned User Requests
ANGARA Statistical Anal. Data
Medical Report Forms

The following user information servicing programs have been written:

REQ1 — program to generate the machine output form for the "health status evaluation card." Two cases are possible: Patient's card printed entirely or only the segment indicated in the request are printed;

REQ2 — batch-mode query program. The program supports retrieval of data based on any conditions specified in the query. The program provides for simple search result processing: Determination of mean value of characteristics, minimum or maximum. When more complex statistical processing is desired, the program can output the data retrieved as a special file. The statistical data processing program can then access the output file and perform the desired statistical processing of the data.

A relational-type language is used to write queries for data retrieval (processing), intended for use by nonprogrammer users. For example, the query "for female workers over 40 years of age, less than 165 cm in height and over 80 kg in weight, determine the mean heart rate after exercise" would be written in this language as follows

```
PAT (P4=2)
INSP (19>40)
AT (AT1<165.0*AT2>80.0)
AVE (H29)
```

The procedure to prepare a query is simplified by the use of a special blank allowing queries to be formalized and decreasing the probability of error, and permitting conversion of queries to machine-readable form directly from the blank.

One shortcoming of batch-mode operation is the need to access supplementary tables describing the names and formats of the fields, and the requirement that the user know the values of the coded parameters. This causes some inconvenience for users with little experience using the system. To eliminate this shortcoming, the MEDIS program for generation and execution of user queries in dialogue mode is provided.

During the course of a dialogue, a user can learn the composition of the data contained in the database, compose a request for data retrieval (processing) in menu mode, inspect the request generated if desired as well as the results of its execution, and store them for printing.

We present below an example of a dialogue with the MEDIS program.

Frame 1:

```
Select the group of characteristics to be used for the search:
    01 — general patient information
    02 — general information on examination
    03 — anthropometry
    04 — case history
    05 — biorhythm characteristics
    06 — social history
    07 — employment characteristics
    08 — biochemistry
    09 — BCG and SCG data
    10 — ECG data, heart rate, blood pressure and cardiac cycle intervals
    11 — results of machine processing of heart rhythm
    12 — information on illnesses
Selection: 01
```

Frame 2:

```
Select the characteristic (or press the return key):
    1 — patient card number
    2 — patient name
    3 — date of birth
    4 — sex
    5 — social position
    6 — occupation
    7 — postal index
    8 — address
Selection: 4
```

Frame 3:

```
Select value for characteristic "sex":
    1 — male
    2 — female
Selection: 2
```

Frame 4:

```
Indicate desired operation:
    1 — compose search conditions including same characteristic (or characteristic of
same group)
    2 — compose search condition including characteristic from another group
    3 — finished composing search condition, go on to position of data processing
(output) request
    4 — cancel query
Selection: 2
```

After the composition of the query request is completed, a verbal description of the query appears on the screen.

Frame 5:

```
As a result of the dialogue, the following query request has been composed (No. 1):
    Find information on the following patients:

        sex — female
        number of children — at least two
    For these patients, find the mean value of the characteristic "age"
    Output to a file for statistical processing the values of the characteristics "age" and
"number of children"
```

If the request has been properly composed, the user gives the program permission to execute the query. The results of the query will appear as follows.

Frame 6:

```
Results of execution of query (No. 1):
    Mean value of characteristic "age" — 42
    Composition of parameters for statistical processing:
        1. Age (80 values)
        2. Number of children (80 values)
```

Statistical data processing package. During execution of a query calling for expanded statistical processing and data analysis, the MASOB software package retrieves and outputs data to a special file, after which the OAD programs access the file and perform the desired statistical processing.

The OAD package is a software system which can construct and develop models of an object based on mathematical-statistical analysis of the information contained in the database. The OAD software package is based on the statistical programs of the BMD and BMDP packages, a number of domestic programs in various modifications, as well as programs developed by the authors. Based on its purpose, the package satisfies the following requirements: Completeness of contents, simplicity of program access, independence of use, modularity, allowing expansion of the package, composite (combined) nature of various methods, availability of service facilities, capability for operation in dialogue mode, orientation toward the nonprogrammer user and ability to utilize databases.

The OAD package is intended to perform typical and rather general tasks arising in medical-biological practice with respect to the apparatus of mathematical statistics. The package implements the following basic classes of programs: Screening and computation of basic statistics, multivariate statistical methods (factor, component, discriminant, canonical and other analysis), regression analysis, analysis of time series, special procedures for data conversion, biomedical methods, processing of consultation data.

Access to the programs of the OAD package is standardized, allowing the use of standard data processing methods in the interactive mode. The job control language used by the package is simple. The user defines the controlling operators of a task in dialogue (or batch) mode, providing a description of the task parameters, selection of the analysis method and program operating mode. All controlling operators are divided into two groups — general and specific. The general operators can be used in all programs and have a standard format. The specific operators are used only by one specific program.

The simple organization of the job control language allows the nonprogrammer user to format tasks by the use of a formalized analysis plan for his application problem. We present below an example of the use of the package in dialogue mode.

Frame 7:

```
Composition of parameters for statistical processing:
    1. Age (80 values)
    2. Number of children (80 values)
Please enter name of statistical analysis program — A02D
HELP — F1                                                  EXIT — F3
```

The output parameters of the OAD package controlling operators are then defined.

Frame 8:

```
PROBLM — task definition operator A02D — scatter diagram
    Number of transgeneration cards (not over 99)                  0
    Number of added variables                                      0
    Print covariation matrix (YES/NO)                            YES
    Print correlation matrix (YES/NO)                            YES
    Number of SELECT cards (not over 99)                           1
    Number of cards with conditional operators
        (not over 9)                                               0
HELP — F1                                                  EXIT — F3
```

Frame 9:

```
SELECT — variable select operator A02D — scatter diagram
     Base variable index                                              1
     Number of cross variables (not over 10)                          1
     Cross variable indices                                   2 0 0 0 0
                                                              0 0 0 0 0
HELP — F1                                                     EXIT — F3
```

The results of execution of the program are printed out or placed in a database for further analysis by other programs of the package. The OAD package relieves the user of routine data analysis operations and requires no knowledge of the techniques of statistical computation. However, interpretation of results, selection of analysis methods and making of decisions are all performed by the user and depend on his experience and qualifications.

Diagnosis program. An example of a real model developed as a result of joint utilization of the database and OAD program package is a model for generating a primary diagnosis (reflecting the level of adaptation capacity of the organism). This model was used to write a program implementing the stop light principle. A green light indicates satisfactory adaptation of the organism to environmental conditions. A yellow light indicates stress on the adaptation mechanism. A red light indicates failure of the adaptation mechanisms.

The additional capabilities of the program include indication of deviations in physiological characteristics typical of each patient, as well as determination of risk factors.

The program automatically generates a primary diagnosis and assists the physician conducting the examination in identifying, with the aid of the machine diagnosis, three groups of patients: Persons whose status of health is within the normal range; persons requiring health improvement or preventive treatment; persons requiring further examination. Depending on the membership of a person in one of these patient groups, specific recommendations are generated for maintenance and improvement of health.

Experience of using the system. The system has been in pilot-scale operation in a number of medical-biological organizations for several years. At present, the largest of the databases in use contains the results of examination of over 3000 patients (10Mbytes). During studies performed using the system for monitoring the environment and public health, informative parameters have been selected, the structure of micropopulations analyzed, functional connections between physiological body system parameters and sex, age, social factors and morbidity have been defined. The results of these investigations have been used to improve the methods of conducting massive preventive screening examinations of the population. In spite of the positive evaluation of medical personnel, a number of shortcomings have been found, inherent in the configuration of the system presently used, particularly the database. They include insufficient flexibility of database logical structure and dictionary (recoding database).

In general, the experience of practical use of the MASOB database indicates the need for a DBMS allowing "easy" modification of the data structure, in response to changes in medical methods, groups of patients, purposes of massive screenings, and also indicates the need for active participation of medical user personnel in the definition of the composition of data, output forms and standard queries. Medical personnel should answer these questions in the process of work with system prototypes in dialogue mode.

The following technology has now been developed for planning databases: Determination of requirements for the database as a result of work by medical user personnel with a system prototype based on the UNISON information retrieval system[4] and industrial implementation of the database using the KVANT DBMS.

The facilities used to create prototypes in the UNISON system include a relational type information-retrieval system with report generating facilities. The system is designed primarily for nonprogrammer users, has flexible database generation facilities, allows the selection of query statistics, editing of the dictionary, its adjustment to a convenient language. However, UNISON has limitation in terms of processing of complex queries and the software reliability is insufficient for the production version.

Operation of a physician with the prototype models all of the basic stages in creating a database: Design of the logical database structure, description of tables, characteristics, connections between them, selection of key fields, loading of the database, construction of a coded value classifier, composition of typical queries, determination of report forms and adjustment of the dictionary. All of these stages are conducted in the course of dialogue with the UNISON system. A HELP system is provided for situations of uncertainty.

During work with the prototype, statistics are kept on queries and characteristics such as the number of types of queries, combinations of different field types in records, field characteristics, selection of descriptor fields, etc. are determined. This information is used by the database administrator to develop the production version of the database.

The production version of the database is implemented using the KVANT DBMS. The KVANT system is a special type of DBMS with inverted file structure allowing the construction of logical database schemas of network, hierarchical and relational type. Databases constructed in the KVANT environment are flexible, can contain large volumes of data and have high reliability and consistency.

Thus, a technology has been developed for planning of databases using prototype systems. The details of the technology which distinguish it from traditional methods include: Participation in planning of the database by end users who are not programmers; implementation of a clearer "requester-developer" communication; planning and creation of the production version of the database by the database administrator based on statistics on queries entered during use of the prototype system. Implementation of this technology provides a high degree of agreement between the database developed and the demands of the end users.

REFERENCES

1. Vorobev, Ye. I., Reznichenko, V. Yu. "The Problem of Creating a System to Protect the Environment and Public Health." Atomnaya energiya, 1981, 50, No. 4, pp. 243-248.

2. "Planning of a Database On the Status of the Environment and Public Health." Ye. I. Vorobev, M. G. Arshinskiy, V. P. Ilin, et. al. Ibid, 1982, 52, No. 3, pp. 147-155.

3. "Application Program Package for Processing and Analysis of Data On the Status of the Environment and Public Health." Ye. I. Vorobev, V. A. Kornelyuk, A. S. Kuzmenko, et. al. Ibid, 1984, 56, No. 1, pp. 43-47.

4. Shreyder, B. D. Organizatsiya poiska i obrabotki dannykh na osnove sistemy UNISON (Organization of Data Retrieval and Processing Under the UNISON System). Moscow, Energoatomizdat Press, 1986, 101 pp.

UDC 681.3.06

Application Program Package for Processing Images On Microcomputers

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian (manuscript received 15 Sep 86) No 3, May–Jun 1988, pp. 98–99

[Article by Ye. P. Anokhina, O. V. Berlyand, T. Yu. Kostyukhina, I. A. Nikishchenkov and O. V. Petropavlovskaya]

[Text] Image processing as a means of improving visual perception and increasing the information content of an object investigated is widely used in space research, medical diagnosis, industrial defectoscopy and scientific research automation. The design of an automated image processing system based on a microcomputer can, on the one hand, expand the area of application by decreasing the cost and size of such a system and, on the other hand, place rigid requirements on system structure and functions. A microcomputer-based automated image processing system is capable only of effective image enhancement, with preference given in selection of processing algorithms to integer arithmetic algorithms, significantly increasing processing speed.

An application program package called "image" [Izobrazheniye] has been developed at Ulyanovsk Polytechnical Institute, intended to improve the quality of half-tone images recorded as a matrix of 256×256 elements with the brightness of each element coded by a single byte. The package is based on dialogue control of image processing; in the development of the package, particular attention was given to organizing the exchange of data with peripheral devices and scaling of mathematical variables in order to prevent overflow of machine words.

The "image" package requires an "Elektronika-60M" microcomputer in its minimum configuration (processor, 8 Kbytes RAM, 16 Kbytes ROM, alphanumeric display and "Elektronika 15 IE 256×256-015" device for display and storage of half-tone images). The display system consists of a half-tone display, three frame memory units, a buffer memory unit and a half-tone display controller, represented in the microcomputer channel as three registers: Status, for input and buffer output. The frame memory units can store and display one half-tone image with each element of the image coded as a single byte and two graphic images with each element coded as a single bit. The buffer memory unit stores a table of amplitude transforms of individual elements, in which the values of brightness are used as an address in a 256×1 byte table. The processor controls the modes of display, data transfer with frame memory and filling of buffer memory by changing the contents of the display device status register. The list of main modules and functions they perform is presented in Tables 1 and 2.

The requirement for minimization of the composition of the system and the rigid limitations of software size and available RAM required the use of assembler in order, on the one hand, to reduce program size and, on the other hand, to decrease system reaction time by efficient organization of the transfer of large data arrays.

The total package size is 16 Kbytes, with 5–6 Kbytes consisting of text information.

Processing time depends on the size of the fragment processed and the processing method. For the maximum size the processing time is: Less than 1 minute to generate a brightness distribution histogram; about 2 minutes for averaging filtration; 5-6 minutes for median filtration algorithms.

Table 1. List of "Image" Software Package Service Programs

Program Name	Functions Performed
History	Fills out and edits patient history based on a form.
Initialization	Fills image recording device control word register in "set up" mode.
Rewrite	Rewrites image from recording device to video memory.
Document	Records alphanumeric information on object studied as input by operator, indicates case history being filled out in bottom quarter of display screen.
A-graph	Generates brightness cross section of image at horizontal section selected by operator.

Table 2. List of "Image" Software Package Processing Programs.

Program Name	Task	Functions Performed
Scale	Image scaling	Enlarges assigned image fragment to indicate scale.
Histogram	Increases contrast	Converts brightness scale by method selected by operator (linearization of brightness distribution histograms; nonlinear histogram transform; stretching of assigned brightness range; equalization of histogram).
Buffer	Elements of cluster analysis	Indicates brightness sections following input of brightness limits by operators.
Filter	Noise suppression	Forms spatial filtering of images by some method (averaging filtration; median filtration).
Contour	Isolation of properties	Isolates contours, suggesting brightness gradient distribution histogram for determination of thresholds.

The "image" package was created as a automated image processing system package to be used as a part of a medical ultrasound computerized tomograph, for which design development had been completed.

UDC 681.3.06

Application Software Package Input and Output of Geometric Information On Patterns

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian (manuscript received 26 Mar 86; after revision 08 Sep 87) No 3, May–Jun 1988, pp. 99–101

[Article by S.V. Naumovich and K. P. Kondrashov]

[Text] **Introduction.** Considerable attention is now being given to the problem of complete automation of preparation, selection of location and spacing of blanks in various branches of the economy. The major stages in preparation of production in light industry include: Design of products, reproduction of patterns in terms of dimensions and height, planning of layout and spacing and actual layout of materials. Complete automation involves automation of all of these stages and their interconnections.

Specialists from the Institute of Machine Building Problems, Ukrainian Academy of Sciences, and the Planning and Design Office of Automation of the Textile Industry have developed and introduced a system for automatic planning of layout and spacing of patterns for knitted products[1, 2].

The initial stage of information processing in this system involves coding and conversion of geometric information on parts participating in the planning process. Usually, special languages are used to describe the geometric information[3, 4]. These languages allow description of relatively simple geometric objects using such concepts as straight-line sectors, circular arcs and curves assigned in canonical form. Furthermore, these languages do not consider the technological specifics of the manufacturing process.

The outlines of knitted product patterns are complex, usually containing a variety of curves, which can only be rather arbitrarily described using the facilities of languages for description of geometric information. An application software package for input and output of geometric information on patterns (the VVGIL package) has been developed for coding and conversion of geometric information concerning such patterns.

Mathematical model. In order to perform geometric planning in automated mode, it is necessary to construct a mathematical model to represent the geometric information concerning knitted product patterns.

Most universal and effective is representation of patterns as polygons, assigned by a sequence of point coordinates. This method allows graphic information input devices to be used to encode the patterns, and also provides the required accuracy, since any continuous smooth curve can be approximated by a broken line with any predetermined accuracy.

An arbitrary pattern in this case can be represented as a set of points

$$T=\{t_i|(x_y,\ y_i)\}_n,\ n\geqslant 4,$$

with the following conditions defined in set T:

$$t_n = t_1, \tag{1}$$

$$t_i \neq t_j, \tag{2}$$

$$L_i \cap L_j = \emptyset, \tag{3}$$

$$|t_{i+1} - t_i| > \epsilon, \tag{4}$$

$$|L_i - t_j| > \epsilon, \forall\, i > j = 1, \ldots, n-1. \tag{5}$$

where $L_i = (t_{i+1}, t_i)$ is a line of the polygon located between points i and $(i+1)$.

Conditions (2), (3) eliminate polygons with intersecting edges (containing double points or intersecting lines). Conditions (4), (5) limit the minimum permissible distances from any point to any line not including the point.

To meet the technological limitations related to the specifics of knitting production, in the coding process each point on the contour (point on the polygon) is assigned a characteristic such as $f(t_{i0})$. This is necessary to isolate in set T the points on pattern curves which are subsets of design points, i.e., points used to reproduce the patterns in different sizes, etc.

Smoothing of pattern curves. The piecewise–linear approximation of a pattern curve means that pattern curves are stored in the computer and represented on graphic devices as broken lines, which does not correspond to the technological limitations as to smoothness of pattern curves. Smoothing of pattern curves refers to the construction of a smooth curve from a broken line.

Suppose we have the broken line

$$r = \{r(t)\}_m \subset T,$$

approximating a pattern curve. We must construct the curve $S(t)$, passing through the points of the broken line

$$S(t_i) = r(t_i)$$

and having a continuous first derivative at all points (including the points fixed on the broken line), except for the beginning and ending points:

$$S'(t_i \emptyset) = S'(t_i - \emptyset).$$

To solve this problem, we use the mathematical apparatus of spline functions[5]. We introduce parametrization on $r(t)$:

$$t_1 = \emptyset,\ t_{i+1} = t_i + |r_{i+1} - r_i|, t = 1, \ldots, m-1.$$

To construct a spline means that in each of the sectors of the broken line we construct a polynomial such as

$$S(t)=At^3+Bt^2+Ct+D,\ t_1 \leqslant t \leqslant t_{III}$$

Knowing the coefficients, A, B, C and D in each of the broken line sections, we can calculate a certain set of spline values at points t_0, $t_0+\Delta t$, $t_0+2\Delta t$, $t_0+3\Delta t$, etc. As a result, when output to the graphic device, the pattern curves are represented as smooth curves.

Reproduction of patterns in dimensions and sizes. The procedure of reproduction of patterns in dimensions and sizes allows us to avoid repeated coding of patterns of different sizes: It is sufficient to have the coordinates of the points on the pattern contour of a set of one size and assign a table of gradations. Computation of the coordinates of patterns of the required sizes is then performed automatically.

The designer sets the reproduction rules for each part of the product. Based on these rules, tables of gradations are generated, containing increments Δx and Δy for the design points to make the transition from size to size.

Pattern reproduction is performed in two stages. At first, the coordinates of the design points are computed:

$$x'_i=x_i+\Delta x,$$
$$y'_i=y_i+\Delta y, \forall\ i\text{-}1, \ldots, n.$$

where x_i and y_i are the coordinates of design points on the pattern for the base size, x'_i, y'_i are the coordinates of design points on the pattern for the next size.

To compute the coordinates of other points, an algorithm is used which is based on the assumption of similarity of pattern contour sectors in the base size and the required size. Suppose we know the coordinates of the design points t_i, t_j, t'_i, t'_j, as well as the coordinates of point t_l in the base size. The coordinates of the point in the required size are then calculated by the equations

$$x'_l = x_i + ((x_l - x_i) \times \cos\varphi + (y_l - y_i) \times \sin\varphi) \times M,$$
$$y'_l = y_i + ((y_l - y_i) \times \cos\varphi - (x_l - x_i) \times \sin\varphi) \times M,$$
$$M = |t_i - t_j| / |t_j - t_i|.$$

Patterns are similarly reproduced for their sizes.

Functional purpose and structure of the package. The VVGIL package is designed for input, storage, conversion and display of geometric information on knitted product patterns. The package implements the following functions:

— Input of information on the set of patterns in a product;

— input of geometric information on each pattern element by means of a semiautomatic coding device;

— computation of the metric characteristics of the patterns;

— display of model patterns on a graphic display screen;

— editing of geometric information in interactive mode by means of a special language;

— generation of gradation tables on the display screen or semiautomatic coding;

— automatic reproduction of patterns in different sizes;

— drawing of patterns on a plotter in actual size or any assigned scale;

— output on request of reference information on patterns and the database status.

Each function is performed by separate modules. The modules are called and loaded by a monitor program using the CHAIN macroinstruction. All functional modules, after they finish operation, return control to the software package monitor. Data are transferred using an area designated in the CHAIN macroinstruction, as well as a virtual array area utilized by the FORTRAN language system.

Method of operation with the package. The VVGIL package allows processing of information on individual patterns, sets of patterns and entire models. A set refers to a selection of patterns required to produce a product in a given size. The complete set of all sizes is referred to as a model.

Operation with the software package is in dialogue mode. The operator selects the desired function from the display keyboard. The monitor calls up the corresponding functional module, which requests data on the model, set, required parts and other necessary information. The dialogue makes wide use of the principle of defaults and suggestions.

Let us study the execution of functions most frequently used to process information on patterns: Input, display and editing of geometric information.

The procedure to input information on product parts begins with a description of the model. The operator inputs the name and article of a model, the scale of sizes, and lists the parts included in the set, as well as their specific details (numbers of parts, requirement for mirror or symmetrical duplication, etc.).

Coding of geometric information is performed using a graphic information coding device, and includes the following steps:

— The code for a pattern and its size are indicated on the display;

— the pattern or its drawing is attached to a plotting board;

— the point type is indicated using the function keys;

— a special pencil is used to read the coordinates of the points.

Paired (such as sleeves) and repeated parts are coded once. Symmetrical parts are coded using half their contour. To code a rectangular part, one need only indicate its dimensions: Length and width. Patterns, the length of which is greater than the operating field of the plotting board, can be input.

When geometric information is output to graphic devices, pattern curves are automatically smoothed. The design points are marked on the curves, and pattern marks are displayed in the central portion (sizes, part names, etc.).

Editing of geometric information on patterns is performed in interactive mode. The results of editing are shown on the graphic display screen. The operator can rotate a part to any angle, change the type of contour points, change the accuracy of pattern approximation, move a point in any desired direction, etc.

Conclusions. The VVGIL package runs under the RAFOS operating system on the SM computers. The minimum hardware configuration is:

— Alphanumeric display;

— type SM-5400 disk drive;

— semiautomatic graphic information encoder (type PKGIO);

— graphic information converter (type UPGI);

— type AP-7251 plotter.

VVGIL Technical Characteristics

Time required to encode one pattern, s	30-60
Time required to draw one pattern, min	Up to 1.5
Maximum number of parts in model	32
Maximum number of points in a pattern	50
Minimum length of coded line sector, mm	3
Number of pattern size characteristics	3
Number of values of each size characteristic	16
Time required to generate gradation tables for one size characteristic, min	2-5

1. "A Set of Programs for Planning of Location and Spacing." L. V. Shagurin, V. Yu. Gudkov, S. V. Naumovich, et. al. Tekstilnaya prom-st, 1985, No. 5, p. 56.

2. Yeshchenko, V. G., Naumovich, S. V. "Technological System for Planning Pattern Layout and Spacing." Ibid, 1986, No. 12, pp. 43-44.

3. Ostafev, V. A., Globa, L. S., Globa, A. V. "Method of Design of Part Description Subsystem In Dialogue Technical Process Automatic Design System." USiM, 1986, No. 2, pp. 112-116.

4. Gorelik, A. G. Avtomatizatsiya inzhenerno-graficheskikh rabot s pomoshchyu EVM (Computerized Automation of Engineering-Graphic Operations). Minsk, Vysheysh shk. Press, 1980, 206 pp.

5. Bayakovskiy, Yu. M., Galaktionov, V. A., Mikhaylova, T. N. Grafor. Graficheskoye rasshireniye FORTRANa (Grafor. Graphic Expansion of FORTRAN). Moscow, Nauka Press, 1985, 288 pp.

UDC 681.3.015:658.018.2

Dialogue System for Engineering Multicriterion Optimization Problems

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian (manuscript received 05 Nov 86; after revision 30 Mar 87) No 3, May–Jun 1988, pp. 101–103

[Article by A. Ya. Oksenenko, Z. Ya. Lure and G. S. Levitin]

[Text] Methods of search for a set of effective (quasioptimal) solutions occupy a particular position in contemporary multicriterion optimization theory[1]. However, recently increasing attention has been given to methods allowing the decision maker to conduct practical selection of parameters of the optimized object most closely meeting the requirements of the task at hand. All such methods are adaptive in nature and are usually implemented as dialogue–mode programs[2].

By adaptive, we mean that such algorithms feature not only adaptation of the search algorithm to the system of preferences of the decision maker, but also and most importantly, adaptation of the decision maker to the capabilities of the object being studied and generation of the concept concerning the properties of the optimal solution.

The method of multicriterion optimization suggested in[3] is based on probing a portion of the N–dimensional space of controlled parameters determined by the equation $X^*_i < x_i < X^{**}_i$, $i=\overline{1,N}$ by means of LP_τ sequences with high distribution uniformity both throughout the entire space and in projections on any subspace.

At each generated point of the sequence, satisfaction of the functional limitations is tested, i.e., the area of permissible values of the parameters is determined. For points satisfying all limitations the values of quality criteria are computed and test tables containing these values are generated for subsequent investigation. This method is a development of the method of random search and does not require the use of a complex mathematical apparatus for its practical implementation. The effectiveness of the method is confirmed by its successful application in a wide variety of technical situations[4-9].

The dialogue system described is based on the use of LP_τ sequences to probe the parameter space and contains a number of functions implementing adaptive methods of optimization and allowing investigation of the object of optimization. A diagram of the dialogue system is shown in the figure. An arbitrary object of investigation should be described in the software modules, allowing fulfillment of the functional limitations to be verified and quality criteria to be calculated for the assigned controlled parameter vector. The initial data prepared by the user include parametric limitations, presented as the minimum and maximum permissible values of each parameter. In the preliminary stage of operation of the system a test table is constructed containing the number of points indicated by the user (a point refers to a specific parameter vector and the corresponding values of criteria). After construction of the test table, the dialogue operating mode is started, in which the command line is interpreted, the function requested is executed and the system awaits the next command line.

The system performs three groups of functions.

Test table processing functions, designed to analyze data:

— Determination of limiting values of the assigned criterion and indication of the number of the corresponding points;

— output of values of the assigned criterion and the corresponding point numbers in the order of decreasing values of the criterion;

— exclusion (temporary) from the test table of all points for which the values of the indicated criterion exceed the set limit (sequential input of limits for each criterion allows the search area to be localized or the optimal solution to be defined);

— calculation of the correlation coefficient of an assigned pair of criteria by a selection indicated in the test table;

— instruction of a projection of the distribution of values of a criterion onto the plane of two indicate criteria;

— exclusion from the test table of all ineffective points (point i is considered ineffective if the table contains a point $j \neq i$, for which $\Phi_{jk} \leqslant \Phi_{ik}$, $k=\overline{1,r}$, where Φ_{jk} is the value of the criterion for point j, r is the number of criteria in the multicriterion problem);

— input of weight factors and determination in the test table of points minimizing the convolution $\sum\limits_k \lambda_k \Phi^*_k$ and the convolution $\max\limits_k \lambda_k \Phi^*_k$, where λ_k are the weight coefficients, Φ^*_k are the normalized values of the criteria:

$$\Phi^*_k = \frac{\Phi_k - \Phi_{k\,\min}}{\Phi_{k\,\max} - \Phi_{k\,\min}}$$

(successive use of this function allows not only location of the optimal solution, but also formation during the dialogue of the weighted responses of the user concerning the relative value of the quality criteria);

— input of the desired values of criteria and search in the test table for points having the minimum euclidean distance from the set point in the space of the criteria;

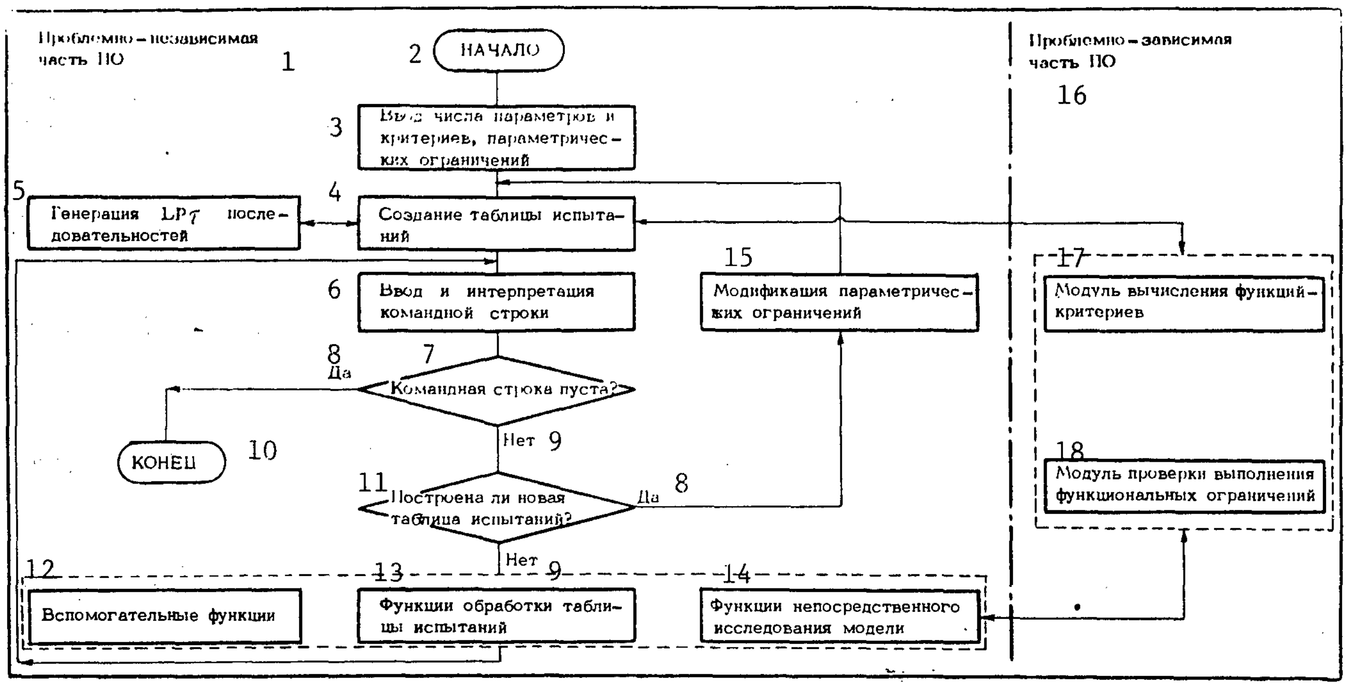
— output of the entire test table or the portion of the table remaining after exclusion of points;

— output of values of criteria and/or parameters of a indicated point;

— clearing of criteria limitations and restoration of the test table.

Since these functions operate on data contained in the created test table, minimum reaction time to user requests is provided, practically eliminating unproductive waiting time.

Direct model investigation functions are designed for graphic representation of equations and make use of the task-oriented program modules. The system constructs graphs of the variation of the criteria indicated by the decision maker as functions of the parameter indicated by the decision maker with values of remaining parameters corresponding to the points, and also constructs lines of the levels of the function indicating the variation of an indicated criterion with two parameters.



Проблемно-независимая часть ПО
1
2 НАЧАЛО
3 Ввод числа параметров и критериев, параметрических ограничений
4 Создание таблицы испытаний
5 Генерация LPτ последовательностей
6 Ввод и интерпретация командной строки
7 Командная строка пуста?
8 Да
КОНЕЦ 10
Нет 9
11 Построена ли новая таблица испытаний?
Да 8
15 Модификация параметрических ограничений
Нет 9
12 Вспомогательные функции
13 Функции обработки таблицы испытаний
14 Функции непосредственного исследования модели
Проблемно-зависимая часть ПО
16
17 Модуль вычисления функций-критериев
18 Модуль проверки выполнения функциональных ограничений

Key: 1, Task-independent portion of software; 2, start; 3, input number of parameters and criteria, parameter limitations; 4, create test table; 5, generate LP_τ sequences; 6, input and interpret command lines; 7, command line empty?; 8, yes; 9, no; 10, end; 11, new test table constructed; 12, supplementary functions; 13, test table processing functions; 14, direct model investigation functions; 15, modification of parameter limitations; 16, task-dependent portion of software; 17, module to compute functional criteria; 18, module to check fulfillment of functional limitations.

The *supplementary functions* include: Input of arbitrary values of parameters or change value of one parameter at an indicated point with subsequent testing of execution at a newly generated point of the conditions of functional limitations and computation of criterion-functions; writing of the test table and other necessary information in long-term storage (during subsequent work sessions with the model, it is not necessary to recreate the test table).

The parameter space probing density can be increased without limit by implementing the procedure which generates a sequence in the vicinity of any indicated point. The area of change of each parameter is reduced by a factor of 10.

Successive application of this function can decrease the area of investigation without limit.

All information in the course of a dialogue can be output to a video terminal or printer in order to document the process of search for the optimal decision.

Utilization of the system has shown that one or two half-hour sessions are sufficient for a user to learn the instructions of the input language and operating principles. This is because of the simplicity and convenience of the instructions in the input language. For example, to calculate the correlation factor between the values of two criteria, the input instruction is $R_{i,j}$, where i and j are the numbers of the corresponding criteria; the values of parameters (criteria) corresponding to point i is executed in response to the instruction TIP (TIK), etc.

It has been found effective to use the system in the stage of developing mathematical models for their preliminary investigation and refinement. In particular, the capabilities of the system to represent information in tabular and graphic form familiar to planning specialists has allowed them to evaluate the adequacy of models and indicate paths for their modification. Over a half year of operation of the system at the Scientific-Production Association of the National Scientific Research Institute for Design of Hydraulic Drives, various models of a radial-piston hydraulic pump with distributing valves, the control system for braking a high-speed positioning mechanism and an electronic control module for the electromagnets of hydraulic apparatus have been investigated and optimized.

The system was developed for SM-4 and SM-1420 computers and can run under OS RV or RAFOS. The system is implemented and models are described in FORTRAN-IV.

REFERENCES

1. Sovremennoye sostoyaniye teorii issledovaniya operatsiy (The Current Status of the Theory of Operations Research). Edited by N. N. Moiseyeva. Moscow, Nauka Press, 1979, 464 pp.

2. Rastrigin, L. A., Eyduk, Ya. Yu. "Adaptive Methods of Multicriterion Optimization." Avtomatika i telemekhanika, 1985, No. 1, pp. 5-26.

3. Sobol, I. M., Statnikov, R. B. Vybor optimalnykh parametrov v zadachakh so mnogimi kriteriyami (Selection of Optimal Parameters In Problems with Multiple Criteria). Moscow, Nauka Press, 1981, 107 pp.

4. "Optimal Planning of Resonant Vibrating Machines." B. I. Kryukov, L. M. Litvin, I. M. Sobol, R. B. Statnikov. Mashinovedeniye, 1980, No. 5, pp. 31-39.

5. Zhitomirskiy, B. Ye., Rubanovich, Yu. A., Filatov, A. A. "Use Of Multicriterion Optimization Method In Planning of Rolling Mill Main Drive Transmissions." Ibid, 1984, No. 1, pp. 33-39.

6. Khomyakov, V. S., Starostina, V. K., Kushnir, M. A. "Multicriterion Optimization of Internal Grinding Heads with Rolling-Surface Bearings." Stanki i instrument, 1984, No. 2, pp. 17-18.

7. Mafter, V. I., Livshits, E. G. "Automated Planning of Electric Winches with Optimal Characteristics." Vesti. mashinostroyeniya, 1985, No. 10, pp. 22–24.

8. Kuritskiy, A. M., Kolman, E. R. "Optimization of Mechanisms for an Automatic Spring Motor Plant." Mashinovedeniye, 1986, No. 1, pp. 27–35.

9. Senkin, Ye. N. "Method of Search for Effective Cutting Tool Designs." Stanki i instrument, 1986, No. 6, pp. 7–9.

UDC 658.012.011.56

Generation of Labor Consumption Standards In an Enterprise Automated Management System

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian (manuscript received 27 Feb 86; after revision 20 Jan 88) No 3, May–Jun 1988, pp. 104–108

[Article by K. F. Yefetova and O. A. Protsenko]

[Text] The present reform in economic management touches all aspects of the production activity of an enterprise. Planning of standards has a decisive role to play.

Standards allow scientific determination of necessary costs, based on which the prices of products must be determined. Improvement of standards generation at an enterprise is among the most important areas for resource conservation in our country.

Particular attention should be given to labor-consumption standards. The labor consumed in making a product is one of the most important characteristics, reflecting a great variety of technical, economic and organizational conditions under which the product is produced. Broader utilization of labor consumption in planning practice and evaluation of the activity of labor teams, the increasing knowledge of labor consumption as a factor in the growth of productivity of labor, require improvement of the method of planning and managing this characteristic.

The problem of labor consumption standards in the manufacture of a product is particularly acute for the solution of optimization problems with respect to generation of the production program of an enterprise. The location of internal reserves and maximum utilization of these reserves as resources remains outside the limits of the problem related to the effect of utilization of reserves, related to the objective prerequisites for changing of the actual labor consumption standards (d_{ij}). Since the problem is solved long before a plan period (March, June, November) begins, the input information used is the labor consumption standard currently in effect.

The problem of decreasing the standard is in practice solved by assigning labor reduction factors for an entire shop[2]. The values of these factors usually vary within limits of 5–12 percent, whereas changes in labor consumption standards for products fall within the range of 2–30 percent.

Therefore, differentiation of standards for products within the limits of a single shop is a necessary condition for improvement of the economic management mechanism and for conversion of shops to independent financing.

Analysis of the factors which determine labor consumption has shown that one means of improving standards is to formalize their definition by the application of statistical methods and modern computer equipment. However, the use of statistical methods in the economics of an enterprise involves a number of difficulties. An important condition for the application of these methods is the availability of a uniform statistical set. In this case, we have in mind the relative, conditional uniformity of the set, meaning that the equations derived are correct for each unit of the set. It is usually considered that the number of observations should be 4–5 times greater than the number of factors analyzed.

The production program of an enterprise, which is the object of investigation in this case, is an extremely nonuniform set. This is most noticeable in enterprises with short-series and individual types of production, where products produced are common only in terms of resources utilized, whereas the structure of these resources used with each product is different. Furthermore, the short "life" of a product (5-7 years) creates additional difficulties in generation of a sufficient number of observations. The problem of sufficiency of observations in this case can be solved by generating uniform products in a group which is periodically manufactured.

Determination of the sufficient number of observations can be undertaken in two stages. In the first stage, all products must be divided into groups in accordance with their functional purpose. This problem is solved in practice by creating functional groups based on professional analysis or the use of the "Industry Product Classifier By Areas Of Technology."

In the second stage, the set of products with the same functional purpose is divided into groups depending on design, technological and economic parameters. The problem arises at this point of determining an efficient number of characteristics to form the operating space of the classification. It can be solved by methods of expert estimation, multivariate classification, group accounting of arguments, etc. Their use in the stage of isolation of essential factors determining labor consumption, as in determination of the degree of influence of some parameter on the labor consumption of a product, has been studied in detail in[1].

The use of expert estimates should be accompanied by a computation of the degree of agreement of various opinions based on the concordation factor and the spearman rank-correlation factor.

The use of expert estimates for each of the factors selected within a functional group results in computation of a variation factor. The variation factors of most groups have been found unsatisfactory. The need has therefore arisen for a multistage classification scheme in which the next stage would consist of subdividing the functional groups based on a set of characteristics, none of which is a necessary or sufficient condition for membership in the group.

One of the methods of multivariate classification, cluster analysis, was selected for this specific implementation. The similarity measure used was the euclidean distance. Analysis was performed on one of the functional groups, including 50 products of a single technological area analyzed on the basis of seven factors. Each product was interpreted as a point in seven-dimensional space. After 49 iterations, 13 homogeneous groups were formed, for only 9 of which the basic statistical characteristics were found to be satisfactory (dispersion, mean-square deviation, variation factor). However, since some of the products were not included in any of the groups and there was a class of anomalous observations, the groups formed were small (three to five products) and ineffective for construction of multifactor regression models to predict labor consumption. Suggestion for the use of the results obtained are presented in detail in[1].

The use of the method of group accounting for arguments, based on the theory of self organization, was attractive due to the absence of rigid limitations as to sample size. However, it was found in the process of implementation that effective use of this method requires rather long dynamic series. Thus, formation of a teaching sample in five years, a testing sample of for the sixth year and a monitoring sample for the seventh year of manufacture of a product yielded satisfactory results. However, there were only three such objects among the 50 products selected.

The results obtained led us to the conclusion that prediction of labor consumption standards within a single enterprise is not expedient, since representativeness of the information used is quite arbitrary. This is because products of a single functional purpose are manufactured at several enterprises in the industry and, therefore, determination of the variation in labor consumption with various factors can be achieved only by analyzing all products of the same functional purpose. The development of industry-wide materials on this problem has allowed the work to be continued. Based on our analysis of industry developments and in accordance with todays requirements for improvement of independent financing within a plant under the conditions of self management and self financing, the task was set forth of creating a dialogue system for the generation of labor consumption standards (called DISFONT).

The guiding principles upon which the creation of the system was based are:

— Assurance of stability of standards over a long period, requiring prediction not only for the planned year, but also for a longer term (2-3 years);

— assurance of conditions necessary for improvement of self financing in shops;

— increased responsibility of workers, for which purpose the assignment as to reduced labor consumption is made at the level of the corresponding subdivisions;

— increased reliability of standards by maximum accounting for nonformalized factors which influence the specific operation of production facilities, for which purpose the algorithm includes a dialogue procedures for correction of calculated values of the standards;

— minimization of waiting time by decision makers;

— assurance of standard development by using industry materials to determine the variation in labor consumption with various factors in accordance with the functional specifics of the various products.

The methodological aspects of the dialogue system were developed utilizing:

— The industry product classifier;

— the industry standard for evaluating technical and organizational levels;

— the industry method for computing supervisors labor;

— the results of statistical processing of analysis for causes of changes in labor consumption over a number of years (at the S. P. Korolev Plant);

— identification of reasons for changes in labor consumption among subdivisions at an enterprise.

The software contained in DISFONT consists of 18 programs, divided into two sets in terms of functional purpose:

— Calculation of technological labor consumption standards;

— calculation of full labor consumption standards (figure).

The set of calculations involved in the technological labor consumption standards consists of three modules: The first determines the standards for labor consumption per unit of product, the second calculates the volume of product in standard hours, the third generates labor consumption reduction assignments for the enterprise subdivisions.

The basic factors considered in determining technological labor consumption included: *a* — the year of manufacture of a product, *b* — the specific share of technically well founded standards, *z* — the level of technological equipment of the product, *d* — the technical level of the facility.

The following calculations are performed to obtain the required information:

— The specific share of technically well founded standards (TOH) in the total volume of labor consumption

$$в = \frac{T_б}{T_{ТОН}} \cdot 100,$$

where $T_б$ is the labor consumption of the product for the various shops in standard hours as of 1 November of the present year (the base); $T_{ТОН}$ is the labor consumption of the product with respect to technically well founded standards;

— the level of technological equipment:

$$z = \frac{П_0}{П_д},$$

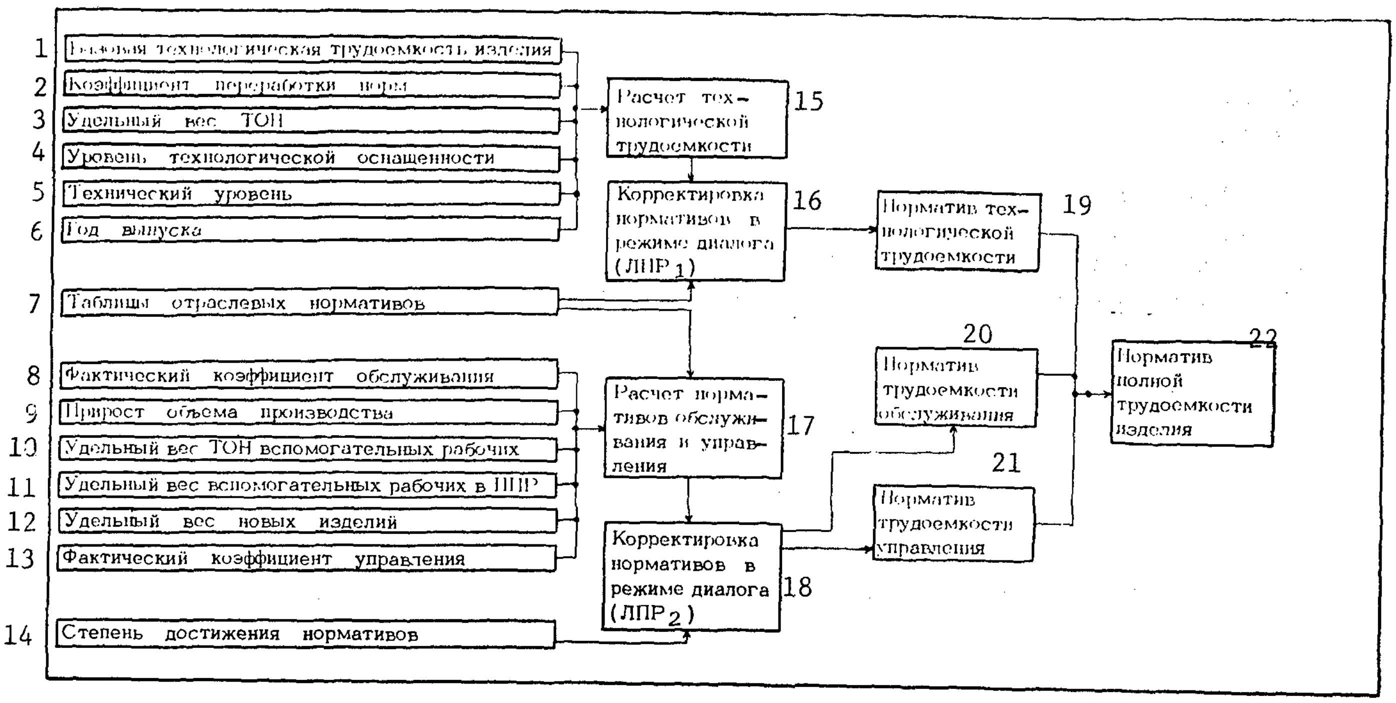
where $П_0$ is the quantity of technological equipment per product in the shop; $П_д$ is the number of parts and assembly units in the shop per product;

— the technical level of production of the product in each shop *(d)* is calculated in arbitrary units, using the industry-standard methods.

Then the values of factors *b, z* and *d* in the industry tables are used to determine the calculated labor consumption factor (K_p), which is then corrected by a correction factor ($K_п$) depending on the year of manufacture of the product *(α)*. If α is 3-7 years, $K_п$=0.8; if $\alpha > 7$, $K_п$=0.7.

The calculated labor consumption (T_p) is determined by the following equation:

$$T_p = T_б [1 - K_п (1 - K_p)].$$


1 Базовая технологическая трудоемкость изделия
2 Коэффициент переработки норм
3 Удельный вес ТОН
4 Уровень технологической оснащенности
5 Технический уровень
6 Год выпуска
7 Таблицы отраслевых нормативов
8 Фактический коэффициент обслуживания
9 Прирост объема производства
10 Удельный вес ТОН вспомогательных рабочих
11 Удельный вес вспомогательных рабочих в ППР
12 Удельный вес новых изделий
13 Фактический коэффициент управления
14 Степень достижения нормативов
15 Расчет технологической трудоемкости
16 Корректировка нормативов в режиме диалога (ЛПР 1)
17 Расчет нормативов обслуживания и управления
18 Корректировка нормативов в режиме диалога (ЛПР 2)
19 Норматив технологической трудоемкости
20 Норматив трудоемкости обслуживания
21 Норматив трудоемкости управления
22 Норматив полной трудоемкости изделия

Formation Of Labor Consumption Standards. Key: 1, Basic technological labor consumption of products; 2, standard revision factor; 3, specific share of TOH; 4, technological equipment level; 5, technical level; 6, year of manufacture; 7, industry standard table; 8, actual servicing factor; 9, increase in volume of production; 10, specific share of TOH of supplementary workers; 11, specific share of supplementary workers among production workers; 12, specific share of new products; 13, actual management factor; 14, degree of achievement of standards; 15, calculation of technological labor consumption; 16, correction of standards in dialogue mode (first decision maker); 17, calculation of servicing and management standards; 18, adjustment of standards in dialogue mode (second decision maker); 19, technological labor consumption standards; 20, servicing labor consumption standards; 21, management labor consumption standard; 22, product full labor consumption standard.

The industry tables present values of the factor *b, d, z* for $K_p=1$, i.e., the best achieved in the industry. The tables are composed by groups of products in accordance with their functional purpose or technical area. Determination of the technological labor consumption standards in the DISFONT package is implemented in two modes: Automatic and dialogue. Automatic mode is implemented in the initial stage of generation of standards and is used to calculate the first version of the production program of an enterprise for the planned year.

In implementing the apparatus of system optimization[4], the criterion function used consists of one or more economic characteristics (profit, material stimulus fund, cost, wage fund, etc.). Maximization (minimization) of these characteristics can result in the appearance of bottlenecks for certain products in terms of utilization of labor resources or equipment. In this case, DISFONT receives information with a list of products for which the standards should be revised. Furthermore, analysis of the growth rate of production volume and productivity characteristics of labor over a five year period may reveal information on products, the labor consumption of which needs to be revised. The person who can make the decision to change the calculated labor consumption standards is the chief of the Labor and Wages Department or his deputy.

The interaction of the user with the system is in dialogue mode in an input message language which is a set of directives. To do this, table 1 is shown on the display screen, indicating the product number, all shops in which the product is manufactured, the basic values of factors determining labor consumption, the best values of the same factors achieved in the industry (for $K_p=1$), the labor consumption, basic and calculated, the year when the computed labor consumption is achieved. The user (decision maker 1) can perform the following actions with the system.

1. Without performing computations, go over to any shop in which the product is manufactured. If all shops have been examined, the instruction is given to go on to the next product.

2. Without performing computations, switch to another product, with subsequent similar actions.

3. Decision maker 1 can edit the calculated labor consumption in two modes: Factor and results modes. Operating in factor mode, the user analyzes the values of the basic and best factors achieved at other enterprises for products of the same functional purpose and factors calculated in a specific shop, and can change the calculated values of one or more factors. The computer calculates and shows on the display the corresponding assigned labor consumption and the year of its achievement. In results mode, the user determines the value of the desired labor consumption, enters it in the computer, and the display screen shows the new computed values of the factors (levels of technological equipment, technical level, specific share of TOH) and the year of achievement of the directed labor consumption.

4. If the user does not accept the results obtained, he can enter an instruction causing the corresponding row of the table to drop downward. Four such shifts will generate a table on the screen consisting of five versions of formation of the various directed labor consumption values. If the desired version is still not seen, the instruction to call up the product is repeated and the search is continued.

Table 1. Generation of Labor Consumption Standards In Dialogue Mode (First Decision Maker)

Product Number	Shop No.	Values of Characteristics												Year	
		Technical Level			Level of Technological Equipment			Specific Share of TOH			Labor Consumption (Standard Hours)				
		Basic	Calc	Direct	Basic	Calc	Direct	Basic	Calc	Direct	Basic	Calc	Direct	Calc	Direct
156	1	0.439	0.496	X	1.4	4	X	60	85	X	460.0	422.0	X	1	3
156	3	0.380	0.510	X	1.68	4.8	X	48	85	X	59.6	42.4	X	1	2
156	8	0.240	0.290	X	6.72	5	X	74	90	X	878.2	860.0	X	1	3
156 per product	Total:	0.510	0.540	X	1.62	4.8	X	54	85	X	4860.0	4640.0	X	1	3

Note: X shows value of characteristic determined by dialogue method.

Table 2. Determination of Standard Specific Share of Numbers of Workers In Dialogue Mode (Second Decision Maker)

Shops	Standard and Actual Specific Shares (%)					Share of Suppl Workers Per TOH	Achievement of Standard (%)				
	ENG	Serv	Gr.serv	Guard	Suppl		ENG	Serv	Gr.serv	Guard	Suppl
Enterprise	15.0	3.0	3.0	1.0	37.0	46.5	120.0	166.6	7.07	100.0	113.5
	18.0	5.0	2.0	1.0	42.0						
Shop No. 1	:XX.X	:XX.X	:XX.X	:XX.X	:XX.X	XX.X	XX.XX	XX.X	XX.X	XX.X	XX.X
	XX.X	XX.X	XX.X	XX.X	XX.X						
Shop No. 2	XX.X	:XX.X	:XX.X	:XX.X	:XX.X	XX.X	XX.X	XX.X	XX.X	XX.X	XX.X
	XX.X	XX.X	XX.X	XX.X	XX.X						
NOTE: Values with ":" on left can be changed by decision maker											

5. Computation of the directed labor consumption by years of manufacture can be performed in automatic or dialogue mode. The first decision maker can assign the decrease in labor consumption by years in standard hours or percent. The absolute values of labor consumption are determined automatically and are shown on the display screen.

The iterative mode of producing the directed labor consumption is continued until the first decision maker is satisfied by the results of the solution. If the user is satisfied with the results, he enters the instruction to store them on magnetic disk. The contents of the screen can also be printed out.

The information generated is sent to the second module, where the volume of production in standard hours is computed for the shops and for the entire enterprise, as well as the percentage of the decrease in labor consumption for the plan period. The results of calculation are sent to the third module, and serve as the bases for the generation of a report entitled "Calculation of Labor Consumption In The Production Program For ... Year Per Shop (Enterprise)."

The necessary information for computations in the third module includes prediction of the reasons for decreased labor consumption and identification of factors for specific subdivisions of the enterprise. The predicted factors result from statistical processing of the file of labor consumption per operation, in which changes in time standards for performance of operations are recorded with an indication of the number of the reason for the change[3]. Analysis of this information over a number of years has led to the following conclusions:

— There is a correlation between the corresponding labor consumption change factors and the years of production of products;

— the structure of the decrease in labor consumption for products in the same year of manufacture differs for different functional groups;

— there is a rather rigid identification among labor consumption decrease factors among the structural subdivisions of an enterprise.

Systematically supplementing and regenerating information on the structure of changes in labor consumption by factors over a number of years, statistically processing the information, one can produce information on the structure of the decrease in labor consumption per product for a year of manufacture in each functional group. It must be possible to correct the structure of changes in labor consumption in dialogue mode in case of previously known measures, causing a change in the technical process or available pool of equipment. Computations performed for one functional group of products and an instrument-building enterprise indicate that the regularities of the decrease in labor consumption established for previous periods can be used as predictions for a subsequent planned period.

Analysis of factors causing changes in labor consumption serves as the basis for development of an identifier of the factor and enterprise subdivision which were the source of the changes. This information creates the prerequisites for making specific assignments for subdivisions to decrease labor consumption for all products, thus improving independent financing and increasing responsibility for decisions made.

The system of computations used to generate standards for total labor consumption consists structurally of three parts: Technological labor consumption, servicing labor consumption and management and total labor consumption. Calculations of decreased standards for servicing and management have no independent value and are used to generate the full characteristic of labor consumption necessary to execute the production program of an enterprise.

The servicing labor consumption per product is:

$$T'_{o}=T'_{o.\text{ф}}\cdot(1-H_{o}),$$

where $T'_{o.\text{ф}}$ is the actual labor consumption involved in servicing of a unit of products; H_o is the standard for decreased service labor consumption. In turn

$$T'_{o.\text{ф}}=T'_{\text{б}}\cdot\frac{T_{o.\text{ф}}}{T_{\text{б}}},$$

where $T'_{\text{б}}$ is the technological labor consumption of a unit of product (the base); $T_{\text{б}}$ is the technological labor consumption of the production program (the base); $T_{o.\text{ф}}$ is the actual servicing labor consumption.

Computation of standards for decreases in servicing labor consumption (H_0) is performed as a function of the following factors: The increase in product volume (ΔB), specific share of supplementary workers in the number of industrial-production workers — N_B, specific share of supplementary workers working according to TOH, N_{TOH}. According to the functional purpose of a product and the absolute values of the factors, the branch tables, interpolated as necessary) are used to determine the approximate value of the standard for decreasing labor consumption (H_{T_0}). The final value of the standard decrease in labor consumption includes a correction of K_K, considering the degree of achievement of the standard level of the number of supplementary workers ($Д_0$) and ΔB).

The value of $Д_0$ is determined in a dialogue procedure, essentially consisting of correction of the standards for the specific share of servicing and management workers by shops, computed by the industry-standard method as the zero approximation. The values of standards in the zero approximation are compared by the decision maker with the actual values at the moment of computation and the corresponding changes are entered through the display to produce acceptable values, based on the specifics of operation of the shops in question (Table 2).

The absolute values of $Д_0$ and ΔB are used to determine K_K in the industry tables. Final determination of the standards for reduction is performed by the equation

$$H_0 = H\delta \cdot K_K.$$

The information obtained is used to determine the standard for servicing a unit of product and the entire production program, and is also the basis for computation of management labor consumption standards.

The standards for management labor consumption are computed by the same method as servicing labor consumption standards. An exception is the algorithm for determination of the actual management labor consumption of a unit of product:

$$T'_{y.\Phi} = T'_б + T'_{o.\Phi} \cdot \frac{T_{y.\Phi}}{T_б + T_{o.\Phi}},$$

as well as the list of factors indicating the influence on the standard for decreased management labor consumption of n, ΔB, N_n, N_{yl}, where N_n is the specific share of production of new products in the production program (per labor consumption); N_{yl} is the specific share of management personnel in category l in the industrial-production personnel.

The result of the computation is the generation of information on the total labor consumption of a unit of product the production program by shops and for the entire enterprise, printed out in four formats. The main users of the results of computation are the Department of Technical Preparation of Production, the Department of Labor and Wages, the Planning Department, the chief economist, and the various shops of the enterprise. The frequency of performance of this task is at least three times per year (during generation of plans in March, June and November), and also when necessary (in case of significant changes to the production-economic activity of the enterprise).

The DISFONT system requires a YeS series (1060) computer, a type YeS7066 or 7927-01 alphanumeric display, a tape drive, printer, and at least 25 cylinders of type 5061 disk storage.

The programs were written in *PL* and assembler. The programs run under OS YeS, version 4.1 or later (*MFT* or *MVT*).

The DISFONT system is standard for enterprises under the Ministry of Communications equipment. The economic effect from its introduction at one enterprise is about 400,000 rubles.

REFERENCES

1. Yefetova, K. F., Zhukov, M. M., Sekirina, L. A. "Some Results of the Application of Statistical Methods to Labor Consumption Prediction Problems." Mekhanizatsiya i avtomatizatsiya upr, 1979, No. 3, pp. 21-24.

2. Danyuk, V. M., Kolot, L. M. "Organization Mechanism of Managing Labor Consumption of Industrial Production." Kiev, 1983-48 pp. (Preprint/Institute of Economics, Ukrainian Academy of Sciences).

3. Yefetova, K. F., Kumkov, Yu. M., Podchasova, T. P. "Automation of Production Planning In the S. P. Korolev Production Association". USiM, 1977, No. 4, pp. 134-138.

4. Volkovich, V. L., Voynolovich, V. M. "Man-Machine Procedure for Solution of Multicriterion Optimization Problems." Ibid, 1979, No. 5, pp. 24-29.

UDC 519.682.6

Simulation of Coal Mine Fire Protection Control Processes

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian (manuscript received 13 Jul 87) No 3, May–Jun 1988, pp. 109–113

[Article by T. P. Maryanovich, A. Z. Naymanov, K. I. Pozdnyakov, M. A. Sakhnyuk, L. G. Khukhlovich and T. N. Yarovitskaya]

{txt}[Text] Problems of preventing and eliminating fires in underground objects are quite pressing and require the development of powerful, reliable, economically effective systems and fire protection equipment, as well as optimal strategies for their application.

Such developments require large numbers of experimental studies in specially equipped test areas (adits, training shafts, etc.).

The existing experimental base cannot fully provide the required volume of field testing, recreate the picture of the entire shaft, consider the specifics of technological processes, or reproduce the entire variety of possible scenarios of the development of fires and extinguishing processes.

Furthermore, field testing requires significant expenditure of time and material resources for preparation and conduct of tests, which are dangerous and may have unpredictable results. Therefore, field testing of existing and suggested fire protection systems and equipment is frequently not desirable or simply impossible.

One of the most effective approaches to expand the capabilities of experimental studies of fire protection systems and equipment is the use of computer equipment and software simulation[1].

The following trends have been developed in recent years in domestic computer simulation practice:

— Creation of universal simulation systems[2-4];

— development of systems for simulation of formalized objects[5, 6];

— development of task-oriented simulation systems[4, 7, 8].

The simulation system for fire protection control processes (SIMPO) developed at the V. M. Glushkov Institute of Cybernetics, Ukrainian Academy of Sciences, in cooperation with "Respirator" National Scientific-Production Association, is a task-oriented simulation system based on universal simulation systems.

The system is designed to perform experiments simulating the processes of development, detection and extinguishing of fires in order to select optimal strategies to protect mines from exogenous fires (arising due to the external causes), considering the cost of fire protection versions studied.

In all stages of the creation of SIMPO (requirements analysis, determination of specifications, planning, coding, debugging, testing and documentation), the developers solved many methodological and technical problems. These problems are general in nature, arising in the development of any task-oriented simulation system, and are of the theoretical and practical interest.

SIMPO development methods and technology. Development of task-oriented simulation systems begins with three interrelated tasks:

— Classification of the concepts, objects, systems and typical problems from the subject area in question based on a detailed analysis (and possibly studies) of the specifics of the subject area, as well as its most pressing problem areas;

— selection (or development) of the conceptual-methodologic basis of the system (the set of basic concepts, plans, methods and principles for representation of data and processes in the subject area);

— selection of a basic programming system.

The conceptual basis of SIMPO was selected in the preplanning research stage, including the following concepts: Shaft, block, working, branch, station, supervisor, extraction face work shift, technological equipment, technological equipment classifier, mine and automatic fire protection equipment, fire protection equipment classifier, topologic graph, combustion system, extinguishing system, extinguishing group, complicating factors, cost parameter catalog, losses due to first, economic effectiveness of fire protection, modeling section, computer experiment, etc.

The following typical tasks were also distinguished: Determination of basic and nonbasic fire protection version functional effectiveness characteristics, distribution of fire protection resources within a mine or region, estimation of prospective fire protection systems and equipment (under development), etc.

The NEDIS system[7] was selected as the basic programming system. The concept of the continuous-discrete system which it contains allows convenience representation of the processes of occurrence, development and extinguishing of fires, as well as the corresponding data structures. This provides a single conceptual basis of description of the subject area and a standard modeling mechanism.

The capabilities of the NEDIS system allow conversion of programming facilities in the subject area (data sets, methods, models, etc.) to standard linguistic elements which can be understood by many users.

The methods and approaches of structured programming[9] were widely used in the development of SIMPO.

The system development process involved all the traditional stages of creation of modern software systems[10].

In the requirements analysis stage, the following problems were addressed:

— Determination of the set of tasks to be performed by the system;

— estimation of the completeness and variety of machine experiments in comparison to experiments in test areas;

— a priori estimation of the cost of the machine experiments, since generation of modeling results with the desired level of reliability requires large numbers of computer experiments;

— interaction of users with the system.

In the stage of determination of specifications, the problems addressed included the functional capabilities of the system, the equipment to be used for machine experiments, the structures and formats of input and output data, types of memory resources used to store the data, and methods to be used to evaluate output variables. The method of data representation and processes to be selected from the subject area were basically determined in this stage.

In this case, representation of combustion processes and the conduct of economic calculations required software implementation of formal mathematical models developed at "Respirator" National Scientific-Production Association[11].

In order to describe all subsystems in a mine and all fire extinguishing processes, it was necessary to develop new simulation models.

A mine was looked upon as a complex hierarchical structure consisting of several standard blocks (sectors). Each block in turn included several workings, workings consisted of branches, branches were divided into stations.

To save RAM and data preparation time, the sets of attributes (parameters) of such subsystems as the working, branch and station were carefully formulated. Problems of tying in the mine-geological and mine-technical parameters, technological equipment, fire protection equipment and mine worker shifts to each of the mine subsystems were solved.

Structure and format of the output data in SIMPO were determined on the basis of the system users' requirements.

The system planning stage begins with the problem of specializing the basic programming system.

Adaptation of the NEDIS system to problems of estimation of the effectiveness of fire protection strategies, systems and equipment was based on the use of the mechanism of library entries and returns (library environment)[7].

The specific usage conditions of the library entries and returns are assigned in the description of the variable portion of the program, called the "insert." The insert is always executed in the context of the corresponding library entries and returns.

The embedding of inserts permissible in the NEDIS language allows the composition of the library environment to be expanded, and permits the formation of a multilevel library environment structure. The higher level of the library environment is accessible to a broader range of users.

The next step is determination of the structure, composition and functional purpose of the entire library environment, as well as its basic components. The basis of solution of this problem is the fact that any simulation program which can run on the computer includes three major components: The model of the effect (process) being studied, the experimental system and various types of data (input, output, etc.). An experimental plan is interpreted as control of an experiment.

In general, a separate plan must be developed for each computer experiment, while one model of an object can be used to implement a number of different experiments. It follows from this that experimental plans, the most frequently changed fragments of simulation programs, should consist of inserts executed in the context of the corresponding library entries and returns (in this case, in the context of the description of models of the objects simulated). The model of an object, in general, is a system of interacting modules, describing various processes. These include the models of a mine and its major subsystems, as well as the models of external actions: Processes of combustion, extinguishing and complicating factors.

Based on an analysis of the structure of modern coal mines, their technological input, fire protection equipment, existing facilities for the formalization of description of the processes and development and extinguishing of fires and classes of tasks performed in this problem area, the library environment is structurally divided into four levels. Any SIMPO simulation program based on these four levels is loaded into the outer most block, the so-called standard environment, for execution.

The standard environment of the NEDIS system includes procedures to control a simulation process, generate a calendar, to order and synchronize processes occurring in parallel), interpretation of many NEDIS language operators with complex semantics.

The fourth level of the library environment includes a description of all typical mine subsystems (blocks, workings, branches, stations, mine workers, the supervisor), technological equipment (conveyor belts, cables, sprinkler systems, etc.), various classifiers and cost catalogs. The descriptions of variables and all types of combustion processes (combustion schemes) are placed in this third level. The second level includes a description of complicating factors, variables and scenarios for extinguishing of fires.

As we have noted, control of the simulation process is executed in accordance with the experimental plan selected in each session. Each modeling session includes a series of experiments. An experiment is a multiple SIMPO model run. A model run represents a single fire.

A modeling session consists of the following stages:

— Input of modeling session initial data;

— generation of the object to be studied (experimental area);

— initialization of the modeling process (generation of external actions);

— running of simulation program and measurement of output data;

— completion of the run;

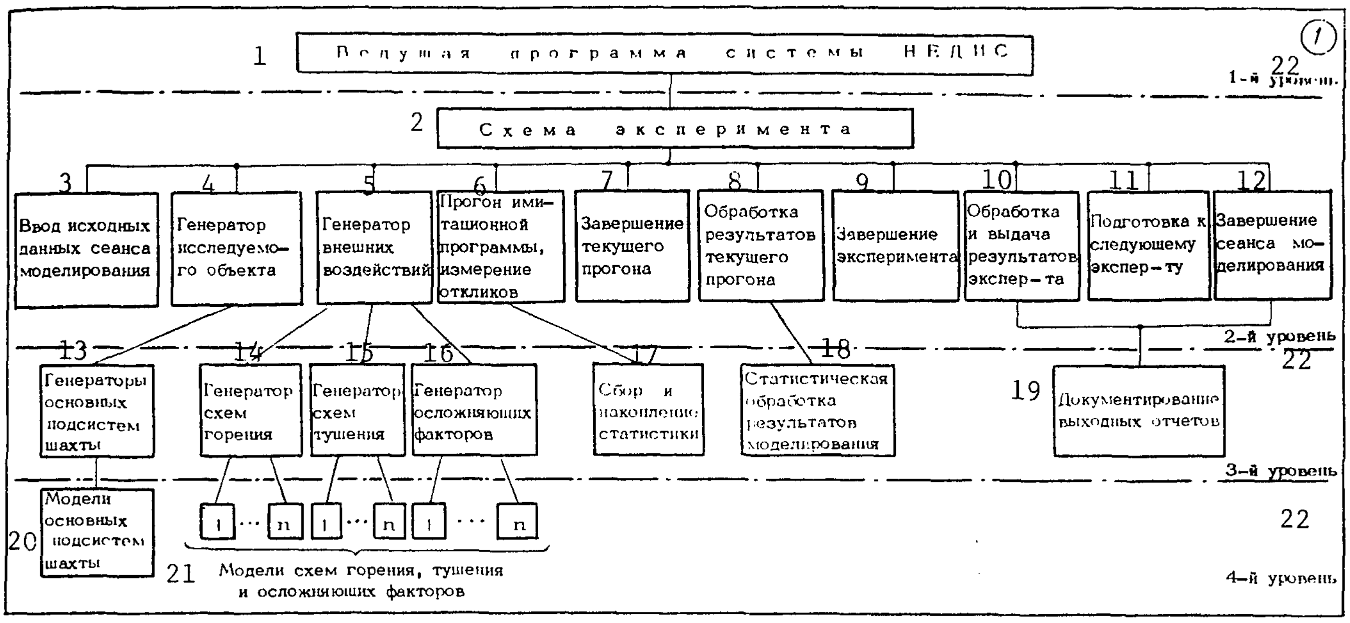
— processing of results from the run;

— completion of the experiment;

— processing and output of results from the experiment;

— preparation for the next experiment;

— completion of modeling session.


1
Ведущая программа системы НЕДИС
(1)
22
1-й уровень
2
Схема эксперимента
3
Ввод исходных данных сеанса моделирования
4
Генератор исследуемого объекта
5
Генератор внешних воздействий
6
Прогон имитационной программы, измерение откликов
7
Завершение текущего прогона
8
Обработка результатов текущего прогона
9
Завершение эксперимента
10
Обработка и выдача результатов экспер-та
11
Подготовка к следующему экспер-ту
12
Завершение сеанса моделирования
2-й уровень
22
13
14
15
16
17
18
Генераторы основных подсистем шахты
Генератор схем горения
Генератор схем тушения
Генератор осложняющих факторов
Сбор и накопление статистики
Статистическая обработка результатов моделирования
19
Документирование выходных отчетов
3-й уровень
20
Модели основных подсистем шахты
1 ··· n
1 ··· n
1 ··· n
21
Модели схем горения, тушения и осложняющих факторов
22
4-й уровень

Key: 1, NEDIS system driver program; 2, experimental plan; 3, modeling session initial data input; 4, studied object generator; 5, external action generator; 6, run of simulation program, measurement of responses; 7, completion of run; 8, processing of run results; 9, completion of experiment; 10, processing and output of experimental results; 11, preparation for next experiment; 12, completion of modeling session; 13, basic mine subsystem generators; 14, combustion system generator; 15, extinguishing system generator; 16, complicating factor generator; 17, collection and accumulation of statistics; 18, statistical processing of modeling results; 19, documentation of output reports; 20, basic mine subsystem models; 21, models of combustion, extinguishing and complicated factors; 22, level *n*.

Thus, a SIMPO experiment consists of a set of modules, each of which supports a stage in a computer simulation experiment (modeling session). The programs in these modules are included in the first level SIMPO library environment, while the program representing the experimental plan, acting as an insert in the library environment, is a chain of operators which call or start the corresponding procedures and classes.

The completeness of a set of experiments which a SIMPO user can run depends essentially on the functional capabilities of and variety of experimental plans (given the composition of the library environment).

For each class of task performed in SIMPO, the corresponding experimental plans have been developed. The determining factors in the classification of the tasks (experimental plans) include:

— The level of the experimental area (entire objects or certain components);

— the composition of the experimental output variables;

— the composition and location of fire protection equipment;

— the fire development scenario;

— the extinguishing tactics and methods;

— the scenario of complicating factors.

The set of experimental plans is determined by the combination (complete or partial) of these factors.

Each experimental plan is invariant relative to the list and sequence of its component modules, but for each class of tasks these modules may differ in their functional purposes.

Thus, the first level library environment contains a set of modules which support the performance of the tasks defined for SIMPO.

Representation of an experimental plan as a chain of calls to the corresponding modules greatly simplifies description of experimental plan programs and expands the range of SIMPO users. Furthermore, with this approach as experience is gained in operation with the system, a set of directives for the SIMPO task language is automatically defined and formulated, supporting the transition to dialogue preparation and generation of programs of simulation experiments.

The planning stage, in accordance with the present technology for development of programming systems, should end in the creation of a hierarchical plan, graphically illustrating the structure of the program and showing the system functions (modules), as well as their subordination.

A diagram of the system hierarchy is shown in Figure 1.

The multilevel structure of this system required that significant attention be given to problems of planning the sequence of development and testing of all modules, as well as preparation of the input data. It should be noted that the structural organization of data in SIMPO depends essentially on the hierarchy of the library environment and the functional capabilities of the experimental plans.

Files are allocated for data (various types of tables, catalogs and numerical characteristics) for all four levels of the library environment. Furthermore, data files are defined to support the functioning of the basic experimental plan modules, particularly such modules as modeling session data input and experimental area generator modules.

The need to determine the economic effectiveness of the fire protection versions studied required the generation of a special file to store the major characteristics (based on modeling results) of the basic and other investigated fire protection versions for all mine workings. Files were also defined for test example data.

As shown in Figure 1, the hierarchy of SIMPO components includes four levels. The driver program of the NEDIS system, located at hierarchical level one is the kernel for all of the modeling programs and is not developed by the user.

With the top-down planning technology, the sequence of development of the modules is generally determined by the corresponding hierarchical plan. Two approaches are most common: Hierarchical (development and testing of module layer by layer) and operational (development and testing of modules on vertical lines as they are called in the operating simulation program).

As SIMPO was created, the need arose to depart somewhat from the top-down technology: Modules were developed from the bottom up on vertical lines, then tested from the top down on vertical lines. The core of each such vertical line consisted of the experimental plan modules.

This approach to planning was demanded by the specifics of the subject area. For example, it does not make sense to develop and test a mine generator if there is no module describing it. "Stubs" were used to represent modules not yet developed.

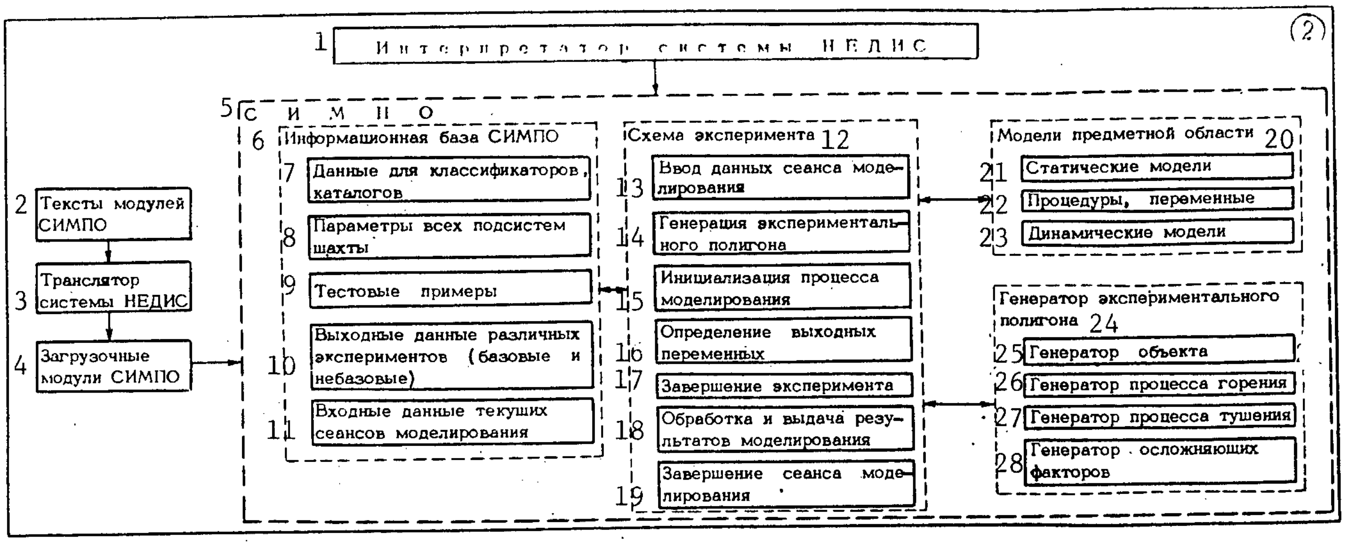
As the system is adapted to the performance of new tasks, it may become necessary to develop new experimental plans. The development of each new plan should begin with verification of the presence of the required modules at all levels of the library environment. If certain modules are missing, the required programs must be developed and introduced to the library environment at the required level.

Functional capabilities of SIMPO. The system is a software modeling system running on the BESM-6 computer, as outlined in Figure 2. The system includes the following components: Subject area models, experimental area generator, experimental plans, and the database.

The component "subject area models" is represented by three library environment levels. The components "experimental area generator" and "experimental plans" are represented in the first level of the library environment. The database contains files to store data for all levels of the library environment, for the experimental area generator, to generate the output data of experiments, test examples, prepare experimental plan programs, modeling results, and input data for various experimental plans.

All modules of the library environment were written in NEDIS and FORTRAN. They are represented using the "procedure" and "class" facilities of the NEDIS language.

The library environment currently contains more than 40 class programs and some 70 procedures. In each SIMPO session, the user must either describe (in NEDIS language) the program for an experimental plan or must use a prepared plan, generally stored in a special file of tested experimental plans.


(2)
1 Интерпретатор системы НЕДИС
2 Тексты модулей СИМПО
3 Транслятор системы НЕДИС
4 Загрузочные модули СИМПО
5 СИМПО
6 Информационная база СИМПО
7 Данные для классификаторов, каталогов
8 Параметры всех подсистем шахты
9 Тестовые примеры
10 Выходные данные различных экспериментов (базовые и небазовые)
11 Входные данные текущих сеансов моделирования
Схема эксперимента 12
13 Ввод данных сеанса моделирования
14 Генерация экспериментального полигона
15 Инициализация процесса моделирования
16 Определение выходных переменных
17 Завершение эксперимента
18 Обработка и выдача результатов моделирования
19 Завершение сеанса моделирования
Модели предметной области 20
21 Статические модели
22 Процедуры, переменные
23 Динамические модели
Генератор экспериментального полигона 24
25 Генератор объекта
26 Генератор процесса горения
27 Генератор процесса тушения
28 Генератор осложняющих факторов

Key: 1, NEDIS system interpreter; 2, SIMPO module texts; 3, NEDIS system translator; 4, SIMPO load modules; 5, SIMPO; 6, SIMPO database; 7, data for classifiers and catalogues; 8, parameters of all mine subsystems; 9, test examples; 10, output data from various experiments (basic and nonbasic); 11, input data for current modeling sessions; 12, experimental plan; 13, input data for modeling session; 14, generation of experimental area; 15, initialization of modeling process; 16, determination of output variables; 17, completion of experiment; 18, processing and output of modeling results; 19, completion of modeling session; 20, subject area models; 21, static models; 22, procedures, variables; 23, dynamic models; 24, experimental area generator; 25, object generator; 26, combustion process generator; 27, extinguishing process generator; 28, complicating factor generator.

The system assignment is formulated using control cards of the NEDIS system, the "Dubna" monitor system and the DISPAK operating system.

Experimental control data contain the following information: Modeling accuracy, probabilities of various types of fires, modeling step, as well as the step length of the processes of combustion and extinguishing, instructions concerning selection of the location, extinguishing plans and equipment, as well as the type of mine worker shift and complicating factor development scenarios.

The system supports the creation of an experimental area by representing the entire object or some of its individual components in each experiment. The experimental area generator consists of a control module and eight functional modules, each of which is designed to create the required number of copies of the corresponding mine subsystems, place technological equipment, fire protection equipment and shifts of mine workers according to the generation assignment list.

Placement of the generation process in a separate module is explained by the fact that generation of the mine information model is not a simple task, since a mine is a complex hierarchical system with many subsystems. For example, one block might contain 5 to 20 workings, each working consists of several branches, each branch includes a large number (dozens to several hundred) of stations.

Initialization of the modeling process, in addition to the usual start-up of the model for computation, also includes start-up of the environment generator, which generates and assigns experimental conditions (nature and specifics of combustion and extinguishing scenario).

Each run of a model is completed by entering the observed quantities on the corresponding histograms or in files, and by computation of the cost of the fire. In the next run, the fire protection equipment is always returned to its initial state, and all mine rescue workers are returned to their positions.

Upon completion of the required number of runs, the modeling results are processed, the cost of fire protection is computed, and possible losses are calculated. If a nonstandard fire protection version is used, the economic effect is calculated. Experiments are completed by printing out a report.

Preparation for the next experiment involves both changing of the placement of fire protection equipment, plus assignment of new values for combustion process parameters, as well as different SIMPO functioning modes.

The output data of an experiment includes: Probability of extinguishing the fire, cost of the fire protection version used, distribution of time when extinguishing efforts were started, distribution of time of completion of extinguishing efforts, distribution of costs for both positive and negative outcomes, distribution of fires extinguished (not extinguished) according to causes, as well as types of extinguishing equipment used.

The output data of a modeling session include the probability of extinguishing the fire for the basic (or nonbasic) fire protection version, costs of fire protection equipment for the basic or nonbasic version, costs of fires extinguished and not extinguished, and economic effectiveness of the various fire protection versions studied.

In the version of SIMPO which has been developed, investigation of development and extinguishing of fires is supported for five types of workings, nine different combustion plans, seven extinguishing scenarios (depending on the type of equipment used), and one type of complicating factor (interruption of water supply).

The total size of SIMPO is about 6,000 NEDIS language operators. The mean time of one experiment is 20-30 minutes for 700-1000 simulation program runs (fire versions), the cost of one experiment is 20 to 50 rubles. Costs for the same number of experimental fires in test areas would be hundreds of thousands of rubles.

The system can be used as a software experimental area for computer testing of existing and prospective fire protection systems and equipment, new fire extinguishing methods and tactics, to search for optimal strategies for protection of mines from exogenous fires. The system can be used to evaluate basic fire protection versions for newly planned coal mines.

All of the major components of the system are open for expansion. Thus, models of the subject area can be supplemented by models of new types of fire protection equipment (such as various automatic extinguishing installations). Naturally, the set of combustion plans will be expanded in the future (for new types of workings), new models of various complicating factors will be developed, and combined mine models will be generated for different regions.

The possibility of representing processes described by differential equations allows automatic programming of ventilation flow control processes, particularly under the extreme conditions of fires.

Expansion of the "experimental plan" component will increase the range of tasks performed with SIMPO. For example, there is a need to develop modules allowing SIMPO to function as an expert system. The system can be used in this manner to solve problems of optimal distribution of limited fire protection equipment within an entire region or in sections of an individual mine considering the potential fire danger present.

Finally, SIMPO can be used to study and evaluate fire extinguishing systems and equipment in other branches of the mining industry and at other objects.

REFERENCES

1. Shannon, R. Imitatsionnoye modelirovaniye sistem — iskusstvo i nauka (System Simulation — Art and Science). Moscow, Mir Press, 1978, 418 pp.

2. Programmnye sredstva modelirovaniya nepreryvno-diskretnykh sistem (Software for Modeling of Continuous-Discrete Systems). V. M. Glushkov, V. V. Gusev, T. P. Maryanovich and M. A. Sakhnyuk. Kiev, Nauk. dumka Press, 1975, 150 pp.

3. Andrianov, A. N., Bychkov, S. P. and Khoroshilov, A. I. Programmirovaniye na yazyke SIMULA-67 (Programming In SIMULA-67). Moscow, Nauka Press, 1985, 307 pp.

4. Litvinov, V. V. Matematicheskoye obespecheniye proyektirovaniya vychislitelnykh sistem i setey (Computer System and Network Planning Software). Kiev, Tekhnika Press, 1982, 176 pp.

5. Krivutsa, V. G. O dialogovom agregativnom modelirovanii slozhnykh sistem (Dialogue Set Modeling of Complex Systems). Kiev, 1982, pp. 3–11. (Preprint, Institute of Cybernetics, USSR Academy of Sciences, No. 82–2).

6. Kalashnikov, V. V., Lutkov, V. M. and Rives, N. Ya, "Organization of Directed Simulation Experiments Based on a Set Simulation System." Teoriya slozhnykh sistem i metody ikh modelirovaniya (Theory of Complex Systems and Methods of Modeling). Moscow, VNIISI Press, 1982, pp. 50–65.

7. Gusev, V. V., Maryanovich, T. P. and Sakhnyuk, M. A. Sistema programmirovaniya NEDIS (The NEDIS Programming System). Kiev, Institute of Cybernetics, Ukrainian Academy of Sciences, 1975, Part 1, 205 pp., Part 2, 185 pp.

8. Voskresenkaya, T. N. "Special Simulation Software." Voprosy modelirovaniya slozhnykh sistem (Problems of Modeling Complex Systems). Kiev, Institute of Cybernetics, Ukrainian Academy of Sciences, 1980, pp. 6–13.

9. Khyuz, J., Michtom, J. Strukturnyy podkhod k programmirovaniyu (A Structured Approach to Programming). Moscow, Mir Press, 1980, 277 pp.

10. Zelkovits, M., Shou, A. and Gennon, J. Printsipy razrabotki programmnogo obespecheniya (Software Development Principles). Moscow, Mir Press, 1982, 365 pp.

11. "Economic Justification of the Creation of New Emergency Mine Protection Equipment." N. M. Khudosovtsev, A. Z. Naymanov, A. S. Avershin and L. G. Khukhlovich. Sposoby i sredstva vedeniya gornospasatelnykh rabot i preduprezhdeniya avariy v shakhtakh, 1979, No. 16, pp. 3-7.

UDC 681.3.065:51

Data Exchange with IK-1 Interface Functional Devices Under OS RV

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian (manuscript received 16 Dec 86) No 3, May-Jun 1988, pp. 114-117

[Article by V. I. Shapiro and L. V. Litver]

[Text] The OS RV real-time operating system is widely used to create automated systems with various purposes and architectures based on the SM computers. Multilevel, multimachine process control and experimental control systems run under OS RV. Communications with computer systems of other architectures are supported by connecting individual communications devices and subsystems to the common bus with 2K, CAMAC, IK-01 and other interfaces.

OS RV provides the programmer with an extensive selection of methods for software implementation of data transfer with special input-output devices. These include direct access to device registers in a task which has the right to access the input-output page, or a task can use the "attach to interrupt vector" system directive or, finally, a special driver can be included as a part of the operating system. The first two versions require the application programmer to have extensive knowledge of the functioning of the operating system, the devices used and the specifics of data transfer with them, requiring careful programming of exchange algorithms each time at the physical level, with a significant portion of the program written in assembler.

The development of a special driver, involving programming of all physical data transfer procedures in assembler just one time, allows input-output programming in application programs to be performed subsequently at the logical level. Application programs can then be written in high-level languages using standard OS RV input-output requests.

As an example of such an implementation, let us analyze a driver written for data transfer with a set of functional load control devices utilizing the IK-1 interface for connection to the SM-4 or SM-1420 control computer through a special interface module (the BIF OSh/IK-1)[1], developed at the Kiev Special Design Bureau for Automatic Control Systems. The system includes a multichannel analog-digital converter, a multifunctional digital-analog element[2, 3], a digital signal input-output element[4], a timer, arbiter, RAM, type KS52.01 microprocessor from the LIUS-2 chip set.

This system forms a functionally complete set of object communications modules, which can be configured and expanded in accordance with the number of monitoring and control channels required for a specific object. The system also includes a universal regulator which does not have a direct IK-1 interface output, but rather is controlled by output signals of the digital-analog multifunctional element. This system of hardware is used to organize control of loaders in agricultural machine resource testing process controller. The loader controller system controls individual modules using either an SM-4 or SM-1420 computer through the interface module, and also independently by means of its own microprocessor. The arbiters are used to organize bus transfers in the loader controller system.

The general principles for organization of physical transfer and control of modules are determined by the characteristics of the IK-1 interface, allowing, when necessary, easy connection of new functional modules to the loader controller system, including standard elements of the LIUS-2 MikroDAT system[3, 5].

This interface utilizes asynchronous signal exchange with acknowledgement, unidirectional bus address lines and control lines and bidirectional data lines. The OSh/IK-1 interface module supports the standard data transfer procedure from the common bus side and generates all signals required for data transfer with IK-1 modules. To convert the common bus signals, the interface module logically includes instruction and status registers, registers for IK-1 address, output to IK-1, input from IK-1, and an interrupt mask register. For the IK-1 modules, the interface module logically appears as a controller supporting generation of address and control signals on the IK-1 bus, controlling interrupts and bus access. In the mode of sequential polling of a group of IK-1 addresses, the interface module can use incremented addressing with automatic address modification after each polling cycle.

Software implementation of transfer in the loader controller system driver also allows simple inclusion of sections supporting operation with new modules.

The driver was developed to be compatible with standard OS RV special input-output device drivers, utilizing standard control structures and some of the operating system supplementary subroutines. Primary attention during development was given to standardization of logical and physical operations, request formats, and the method of configuring the driver for a specific loader controller system element configuration.

When input-output operations are executed, the driver utilizes static and dynamic information stored in the system data structures. The data structure for the loader controller system driver were developed according to the system standards. Static information stored in the driver includes the module type code, address on IK-1 bus, interrupt level for an initiative module, IK-1 bus number, module number, as well as additional information specific for each module. For example, the RAM module includes the address field, the timer module includes the initial interrupt signal generation interval, etc. Dynamic information such as the current addresses of arrays, counters, channel numbers, etc., is generated by the driver as it processes a request for a transfer operation. The system data structures include one DCB (device control block) for each loader controller system and one UCB (device controller) and SCB (controller status) for each module. All modules included in the loader controller system have a single logical name KUxx in OS RV, where xx is the ordinal number of the module. Agreement between the ordinal number of the module and its type is established when the driver is included in the system.

The procedure for inclusion of a loader controller system driver in the operating system is executed in dialog mode and consists of the following stages:

— Input of information on types, addresses and number of modules included in the loader controller system, and generation of the configuration description file KUPAR.MAC;

— translation of the driver text and basic data structures together with the file KUPAR.MAC;

— instruction of the loader controller system driver — creation of the file KUDRV.TSK and KUDRV.STB;

— loading of the driver.

A command file in the system processor input language for indirect command files (KUGEN.CMD) has been developed to execute these stages. The loader controller system driver is loadable, which must be considered in generating a specific operating system version within which it will function. The direct driver loading procedure can be executed during initial OS RV start up using an instruction in the start up command file.

Modification of the loader controller system configuration (types, addresses or number of modules) requires modification of the driver by the procedure described above.

The driver is constructed of standard section: To start input–output operations, process interrupts caused by power supply failure, interrupts from modules, to process time–outs.

A power failure interrupt is imitated upon system start up and when the driver is loaded. This allows this section to perform all actions involved in initializing the driver data structures and initial setting of modes in the modules. A interface module status check is performed, IK–1 interrupts are disabled to eliminate false operations, the microprocessor is started and executes the false instruction to jump to address 0.

The interrupt processing section recognizes the type and address of the module generating an interrupt request, and sets the corresponding interrupt masks. An intrasystem request to start the tasks attached to the interrupts of the module involved is then generated.

All loader controller system modules operate with no wait states. This avoids processing of a time out when a transfer operation is performed, and the time out processing section in the driver consists only of an immediate return to the main program.

Requests to start input–output operations are processed by special driver sections for each module and each operation type. We should note that many requests do not require actual execution of physical transfer operations with the module and can be executed regardless of other requests. They include the operation of producing parameters of a module (type, address, interrupt level), shifting a module to "independent" status, in which all requests to the module are blocked by software, reading of digital signal input–output element interrupt data, determination of the time parameters of the timer.

The basic analog and digital input–output operations include input–output of information for a sequential group of channels of a module and input–output according to an arbitrary list of channel numbers. The group of initiative operations supports configuration of the driver to operate with digital signal input–output elements, the timer and microprocessor in initiative mode. When an interrupt signal is generated by any of these modules, one or more tasks is started, performing the required interrupt processing operations.

The driver supports the organization of cyclical start up of operations in the analog–digital multichannel converter and digital–analog multifunctional element at fixed time intervals using the loader controller system timer, which can be programmed for any time interval from a few hundred microseconds to minutes and hours. The multichannel analog-digital converter supports cyclical polling of groups of channels at fixed addresses with sequential filling of a fixed-length data array. An application program must output a single request to execute a cyclical operation, indicating as a parameter the number of polling cycles. Upon completion of the polling operation, the driver utilizes a standard, OS RV procedure to transfer information indicating normal completion of the operation to the program. The operation of cyclical output of signals to fixed digital-analog multifunctional element channels operating in digital-analog converter mode is performed similarly. When the digital-analog element is operating as an interpolater, repeated processing of an array of values of the function to be interpolated is supported, outputting signals at fixed time intervals.

Software control of the arbiters is used to control modules on any bus. The driver supports programmed arbiter operation to organize communications among selected buses within a block. Transfer can be performed directly with the modules in direct bus access (bus capture) and time sharing modes with active elements (such as microprocessors) working on the bus. When a logical request is formed for data transfer with a module, it need not be located on any particular bus within the block. Verification of the physical address of the module on a specific bus within the block and organization of transfer with the module through the proper arbiters are performed within the driver using the same transfer algorithm.

When a microprocessor is included in the loader controller system, some of the low-level load-mode monitoring and control functions can be performed using the microprocessor and RAM. This requires communications between the microprocessor and the SM-4 (or SM-1420) processor for mutual exchange of status signals and required actions. A certain RAM location is selected with an arbitrary but predetermined and fixed address, and used to store information on the microprocessor status. The distribution of specific bits of the location for coding of the various status signals is performed as the system is programmed and is arbitrary. The signals are generated by the microprocessor as the program is processed.

Upon detection of certain logical conditions, module or RAM statuses, etc., the microprocessor, by generating a specific status word, can send an interrupt signal to the SM-4 processor, starting a special system task. This task can in turn read the microprocessor status (i.e., the value stored in the RAM location allocated for storage of the status), analyze the situation and take the necessary action. The loader controller system driver, in turn, allows an OS RV task to generate an interrupt signal for the microprocessor to start a special program loaded in the RAM module.

When operating as a part of the loader controller system, a RAM module can be used to store programs to be executed by the microprocessor as well as associated data — load tables, coefficients, module polling results. The volume of RAM and distribution of addresses are determined by the structural configuration of the loader controller system considering the availability of arbiters. Since communications between different IK-1 buses in the block are organized using the 15th bit of the address bus, the maximum RAM which can be used on a bus with an arbiter is 32 Kbytes. Exchange with a RAM module is performed by direct bus access or time sharing with the microprocessor, as determined by a special parameter set in the request for the read or write operation.

The microprocessor can execute a special operation which writes an array of data into RAM and then starts the microprocessor.

Let us study the processing of requests to start input–output operations requiring physical access to the modules in somewhat greater detail. The following initial assumptions were made in the development of these sections:

— In the initial state, all interrupt levels to the IK–1 are masked, the masks are removed at the corresponding levels as requests for initiative operations for a module are processed;

— in the interrupt system on the common bus, the loader controller system is connected at level IR5;

— in the initial state, the interface module is programmed for operation with input–output devices, the mask blocking interrupts from the IK–1 and mask blocking requests for bus access are not set (BAB Const);

— after execution of any transfer operation with input–output devices or RAM, the operating mode of the interface module is restored.

The general procedure for a physical transfer with the modules is standardized and consists of the following functional operations:

— Programming of arbiters to establish communications with the internal IK–1 bus on which the module is installed;

— switching of the processor to the sixth interrupt level (blocking interrupts from input–output devices);

— programming of the interface module for the assigned operating mode — permitting incremental addressing, setting the interrupt mask, setting the transfer mode with input–output devices or RAM, etc. (selectively depending on the trânsfer conditions required for a specific module);

— sending of the IK–1 module address for communications to the interface module;

— reception or transmission of databytes (transfer of information with the module) with automatic channel address incrementing;

— clearing of the arbiter (breaking of communications with the block bus);

— restoration of the initial interface module mode;

— processor interrupt enable.

Requests usually use two buffers: A buffer for the input–output of data codes and a buffer containing control information (addresses of the channels pôlled), and therefore the direct transmission of information to or from memory utilizes the special operating system subroutines ⊛GTWRD, ⊛PTWRD and ⊛GTCWD. These subroutines are present in the operating system only in special cases. Therefore, for normal operation of the driver the texts of these subroutines are contained in the main driver text, but they are compiled and included in the driver object module only if not present in the operating system. This condition is tested by the macroassembler by determining the values of the corresponding system global identifiers in the file RSXMC.MAC.

The queue of requests to the module as an individual system device is used to resolve collisions occurring when the same module is accessed by different tasks. However, this mechanism is insufficient for cases of simultaneous accessing of different modules, which is not permissible, since all transfers are performed through the interface module registers. To eliminate problems with such accesses, processor interrupts are disabled for the time required to execute a physical transfer instruction.

Correctness of execution of these operations by the driver can be verified in the task which generates an input–output request by analyzing the operation completion code sent by the driver to the input–output status block. The values of completion codes correspond to the OS RV standards, and can be used in assembler programs as numerical or symbolic values. The major ones are as follows:

— IE.BAD — Error in request parameters or impermissible module type;

— IE.IFC — impermissible operation code;

— IE.PRI — violation of privileges;

— IE.FLN — module transfer to "independent" mode;

— IE.SPC — impermissible address of channel number list array;

— IE.NST — task not found in system catalog;

— IE.NOD — no free dynamic RAM for creation of request processing element;

— IE.CON — reattachment of task not permissible.

The range of input–output function code values for the loader controller system driver and their mnemonics were selected so that they differ from the corresponding codes for standard input–output device drivers. When tasks are programmed in FORTRAN, octal codes are used, while macroassembler programs can use mnemonic symbolic references. For this purpose, a special module called KUFCN.MAC has been developed, in which the symbolic code and their corresponding numerical values are entered as global variables. After compiling the module KUFCN.OBJ can be included in the OS RV system library — SYSLIB.OLB, allowing computation of the values of symbolic variables corresponding to input–output function codes for the loader controller system in the task construction stage utilizing these codes.

As an example, we present several values and symbolic representations of input–output function codes:

— IO.AIG=16,000$_8$ — input of analog information with successive channel polling;

— IO.AIL=16,001$_8$ — input of analog information using a list of channels;

— IO.CAI=16,005$_8$ — cyclical analog-digital converter polling;

— IO.AOG=17,000$_8$ — output of group analog signals to multifunctional digital-analog element — digital-analog converter;

— IO.TIT=15,002$_8$ — attach task to timer interrupts;

— IO.TTA-15,003$_8$ — release task from timer interrupts;

— IO.WRM=17,406$_8$ — store array in RAM.

In all, processing of 34 different types of input-output requests is supported in the driver.

The limiting requirements for tasks operating with the loader controller system driver are related to the need to perform long, multiple-cycle accesses and store data in user buffers, change device operating modes, etc. Thus, output of a request to shift a module to "independent" mode is possible only in a privileged task. A task attached to interrupts by the microprocessor, timer or digital signal input-output element must be constructed as resident and must be stored in memory before it is started. Tasks starting cyclical operations in the analog-digital and digital-analog converter elements must have similar characteristics. In general, these limitations are not too rigid and help to improve the operational stability of the software.

The effectiveness of the use of the loader controller system driver has been confirmed by its operation as a part of an agricultural machinery resource testing process controller, including for the new "Don-1500" high-productivity combines.

REFERENCES

1. OST 25.984-82. GSP. "Hardware System for Local Information-Control Systems (KTS LIUS-2). IK-1 Interface, Technical Requirements." Dated 01/01/83.

2. Zhabeyev, V. P., and Krotevich, V. A. "Multichannel Linear Interpolator for Programmed Movement Control Systems." Mekhanizatsiya i avtomatizatsiya upr, 1981, No. 4, pp. 38-39.

3. Author's certificate 840957 USSR, MKI^3 G06 G7/26. "Multichannel Functional Converter." V. P. Zhabeyev, L. G. Zyabrev, and Ye. M. Kolman, et. al. Otkrytiya. Izobreteniya, 1981, No. 23, pp. 43-44.

4. Author's certificate 830367 USSR, MKI^3 G06 F3/04. "Device for Interfacing Computers with Discrete Sensors." V. P. Zhabeyev, V. V. Kalinichenko and V. I. Korolkevich, et. al. Ibid, No. 18, pp. 127.

5. Seriyno vypuskayemoye i perspektivnoye oborudovaniye. Otraslevoy katalog GSP (Series Produced, Promising Equipment. Catalog of State System of Instruments). Moscow, TsNIITEIpriborostroyeniya Press, 1984, No. 5, 80 pp.

Manufacturing Quality Assurance Under The New Economic Conditions

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian No 3, May-Jun 1988, pp. 118- 119

[Article by V. S. Dubovik]

[Text] A zonal scientific- practical conference on "technological manufacturing control systems and product quality in the light of the resolutions of the 27th CPSU congress" was held in Penza 10- 11 December 1987. It was organized by the Scientific Council On The Problem Of "Cybernetics" of the Ukrainian Academy of Sciences, the V. M. Glushkov Institute of Cybernetics, Ukrainian Academy of Sciences, the National Scientific Research Institute of Standardization, USSR State Standards Commission, the Volga House of Scientific and Technical Propaganda, the Committee on Standardization, Quality and Metrology, Ukrainian Scientific and Technical Society, and other organizations in order to discuss the status of affairs, evaluate and share leading scientific and practical experience in the area of production management and product quality.

The conference involved the participation of scientists and specialists, leaders of industrial enterprises and associations, technical testing department chiefs, product quality management service chiefs, leading specialists from organizations and enterprises of Voronezh, Gorkiy, Kiev, Leningrad, Moscow, Penza, Sevastopol, Ulyanovsk and other cities.

Opening the conference, the scientific director **P. Ya. Kalita** (Institute of Cybernetics, Ukrainian Academy of Sciences) noted that as enterprises in our country make the transition to operation with full self financing and cost accounting, as the law on state enterprises (associations) takes effect, during the period of continued perestroika, the problem of increasing the effectiveness of operation of enterprises and organizations, improving product quality and competitiveness will move to the four.

The report of **V. I. Tobis** (National Scientific Research Institute of Standards, USSR State Standards Commission) noted that improvement of product quality is most closely related to the results of functioning of the product quality management system. Analysis of the state of affairs with respect to the operation of such systems in the ministries (departments), at enterprises and in organizations has shown that no essential improvement in product quality has resulted from their introduction. The institute is therefore now developing the basic concepts for product quality management under the new economic conditions.

There was great interest in the reports of P. Ya. Kalita, demonstrating the need to improve product quality management systems at enterprises and in organizations, discussing scientific- methodologic and practical problems of implementation of structured goal- oriented management methods using the example of an organizational- technological decision- making system encouraging stability in quality formation processes (STK PRIROST- PK). Discussing problems of design and functioning in detail, the speaker conveniencly demonstrated the effectiveness of STK PRIROST- PK as a means for increasing product quality at various management levels.

Problems of the development and use of software to implement diagnostic procedures in management of production and product quality management were discussed by **Ya. B. Malanyuk** and **B. F. Opara** (Institute of Cybernetics, Ukrainian Academy of Sciences).

The practical experience of applying STK PRIROST- PK to management of a fish canning enterprise was discussed by **L. I. Velichko** ("Atlantica" Fishing Industry Production Association, Sevastopol).

The reports of **V. S. Dubovika** (Special Design Bureau MMS [expansion not given], Institute of Cybernetics, Ukrainian Academy of Sciences) and **N. V. Konovalov** (Institute of Super- Hard Materials, Ukrainian Academy of Sciences) discussed an approach to practical problems of quality management of developments based on STK PRIROST- PK in a two- step "design- experimental production" system, operating within the framework of an academic scientific- technical complex.

M. I. Kulagina (Central Scientific Research Institute of Wool Products, USSR Light Industry Ministry, Moscow) informed the participants of the purpose, composition and functioning sequence of the INZHENER production process operational control system, based on methods of situational analysis implemented on computers.

The problems of production management to assure manufacture of products with the highest quality ratings were discussed in the reports of **V. V. Lazebnaya** and **N. V. Stolyarenko** (Institute of Cybernetics, Ukrainian Academy of Sciences).

A. P. Lagoda (Institute of Cybernetics, Ukrainian Academy of Sciences) discussed an approach to improvement of product quality in machine building by development and improvement of designs in the experimental production stage.

The participants of the conference noted that the experience gained in the employment of product quality management systems at industrial enterprises has demonstrated the need for significant improvements in their effectiveness. This task is particularly pressing given the transformation of enterprises and organizations to full cost accounting and self financing, the introduction of state acceptance testing of products, and the affirmation of the 9000 series "Quality Assurance System" international standards, observation of which is an important factor in the improvement of competitiveness of domestic products on the world market.

A number of USSR state standard commission standards documents on problems of development and introduction of product quality management systems have been changed in response to adoption of the USSR law on state enterprises (associations); therefore, the enterprises themselves must activate the creation of effective product quality management systems, capable of achieving a radical improvement in product quality and increasing production effectiveness. At the same time, the enterprises are experiencing an acute need for improved scientific- methodological support, effective standard plan decisions and the means necessary to create such management systems. The concept of product quality management, developed by the National Scientific Research Institute of Standards, USSR State Standards Commission with the assistance of leading specialists from the ministries and departments, reflecting the specifics of the transformation of the economy to the new economic conditions, should help solve these problems.

Many scientific research institutes are working on the development of improvements to product quality management systems, the use of which can even now assure the implementation of a number of the international series 9000 standards and improve the quality of products produced and the technical- economic characteristics of enterprises. These include primarily developments related to planning of organizational- technological production and product quality management systems, based on STK PRIROST- PK (Institute of Cybernetics, Ukrainian Academy of Sciences), as well as the INZHNER system (Central Scientific Research Institute of Wool Products, USSR Light Industry Ministry).

In the exchange of opinions, the participants of the conference recommended that enterprises (associations) and organizations activate the creation of highly effective product quality management systems, basing them on economic management methods, corresponding to the new economic conditions. The creation of such systems should be oriented toward the internal series 9000 "quality assurance system" standards and should make broader use of leading scientific achievements and the practical experience which has been gained in the area of improving product quality management systems.

The basic concepts of product quality management under the new economic conditions suggested by the National Scientific Research Institute of Standards, USSR State Standards Commission, and the developments related to the creation of STK PRIROST- PK and the INZHENER system were applauded. It was decided that development of work on planning of organizational- technological production and product quality management systems and the development by the National Scientific Research Institute of Standards, USSR State Standards Commission, of methodological recommendations for organization of the work performed at product quality management organizations and services in the ministries (departments), organizations and enterprises should be continued, reflecting the specifics of this work under the new economic conditions.

Management of Data Processing Technology In Computer Centers and Computer Networks

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian No 3, May-Jun 1988, pp 119-121

[Article by B. N. Panshin]

[Text] The seminar "Problems Of Management Of Data Processing Technology At Computer Centers And On Computer Networks," organized by the Republic House of Economical and Scientific-Technical Propaganda, Institute of Cybernetics imeni V. M. Lushkov, Ukrainian Academy of Sciences, and the exhibition of achievements of the Ukrainian National Economy, was held 2-3 September 1987 in Kiev.

Following the tradition (this seminar has been held each year in September since 1978), the seminar analyzed problems related to the creation, development and operation of organizational-technical and programming facilities for management of data processing technology at a computer center, in local and distributed computer networks. It was noted that under today's conditions, these problems must be studied from the standpoint of the methodological concepts of the new information technology.

The reports presented at the seminar were on the following main themes:

— Methodology of planning and improvement of technological data processing systems;

— development and introduction of new algorithms and software facilities for managing data processing technology;

— development of facilities to support friendly interfaces for computer system and users.

The report of Ye. M. Lavrishcheva (Special Design and Technological Bureau, Production Association, Institute of Cybernetics, Ukrainian Academy of Sciences, Kiev), **"Principle of technological preparation of development of data processing application programs"** analyzed the major tasks in technological preparation of the development of application programs, concluding assurance of software package ease of use and quality, development of technological processes and lines based on technological and organizational support. To solve these problems, the author suggests and provides a foundation for the concept of metatechnology[1], allowing methods and facilities for the development of application programs required for the subject area to be represented as chains of actions by technological processes and lines, performed on a system of conceptual and technological objects. The structure of the elements of the metatechnology is presented as applicable to the class of TSOI [expansion not given] problems.

[1]Lavrishcheva, Ye. M. Osnovy tekhnologicheskoy poodgotovki razrabotki prikladnykh programm SOD (Principles of Technological Preparations For Development Of Data Processing System Application Programs). Kiev, 1987, 30 pp. (Preprint, Institute of Cybernetics imeni V. M. Glushkov, Ukrainian Academy of Sciences: 87-5).

The report of Ye. B. Bibik, N. G. Varavina (Institute of Cybernetics, Ukrainian Academy of Sciences, Kiev), A. D. Savchenko (Special Design and Technological Bureau, Production Association, Institute of Cybernetics, Ukrainian Academy of Sciences, Kiev), **"Improvement of the organizational structure of a computer center."** discussed the activity of a computer center as a set of interrelated functions forming a hierarchical structure. The tool used to select the specialization of computer center subdivisions by the authors consisted of structural matrices and cluster analysis. The production structure of a computer center formulated as a result of these developments is presented[2].

The report of O. P. Kirilenko, (National Scientific Research Institute of the Organization of Production and Labor In Ferrous Metallurgy, Kharkov), "Structural-functional model of the technological process of data processing" analyzed a mathematical model of the technological process of information processing, the distinguishing features of which include: Approximation of the concepts of the subject area system and the conceptual system of the model; inclusion of a stage of planning computations in servicing requests for information-computer operations; inclusion in the model of previously developed support programs.

The report of G. F. Yanykh, **"Problems of creation of an automated computing process management system in a multimachine computer center,"** (Central Scientific Research Institute of Automated Management Systems, Riga) was dedicated to specifics of managing the computing process in combined specialized computer systems. The speaker reported a system for organizing the computing process developed at his institute. In the present stage, software facilities are being developed to compose a weekly schedule for allocation of machine resources with a detailed list of tasks and considering assigned schedules for producing computation results.

The report of N. D. Moshkina, **"A system for simulation of discrete-continuous processes"** (Moscow Engineering and Physics Institute, Moscow) discussed an original simulation system which has been developed and is now in practical use, distinguished by its great speed and well developed linguistic facility; the system can simulate the behavior of complex systems both in time and space; a single time scale is used to model both discrete and continuous processes; object modules are small (5-6 times smaller than for the widely known GPSS system).

The report of A. A. Yemelyanov, **"The status of DEMOS family operating systems,"** (Moscow Engineering and Physics Institute, Moscow), was dedicated to the analysis of the development of operating systems based on the Unix. standard. The major advantages of such operating systems include: Possibility of organizing interprocess communications channels; development of hardware facilities allowing rapid creation of new dialog information systems; development of program debugging facilities; the ability to create new compiles (availability of compiler generators). The speaker briefly described commercial versions of Unix.-type

[2]Sovershenstvovaniye organizatsionnoy struktury vychislitelnogo tsentra (Improvement Of The Organizational Structure Of A Computer Center). Ye. B. Bibik, N. B. Varavina, B. N. Panshin, and A. D. Savchenko. Kiev, 1987, 23 pp. (Preprint, Institute of Cybernetics imeni V. M. Glushkov, Ukrainian Academy of Sciences: 87-39).

operating systems now available: The DEMOS OS for the YeS and SM computers and several types of microcomputers. The RUBIN DBMS is available for use on the SM computers.

The report of N. A. Gudzenko, **"Technology of Planning Mobile Distributed DBMSs"** (Institute of Cybernetics, Ukrainian Academy of Sciences, Kiev) was dedicated to practical testing of a technology for using the capabilities of the C language and the system library of the DEMOS OS to develop specific data management systems. The actual data base in implemented as a set of fixed-length bytes (each set consists of a key and a record). The kernel of the system (i.e., the *DBM* library) can be transferred to any computer. The *DBM* can construct distributed DBMSs, the *DBM* being easy to modify for specific operating conditions. This software package for data management can be easily and conveniently debugged under the control of the DEMOS OS using the *CDEB* debugger, then can be transferred to computers of another type.

Problems of creating systems for management of the computing process were also discussed in reports of V. A. Ubogov "Implementation of local 'star' type computer networks under OS RV and RAFOS" (Chelyabinsk), V. N. Grishchenko, V. I. Sumkin "Facilities for data transmission in organization management systems: (Institute of Cybernetics, Ukrainian Academy of Sciences, Kiev), A. R. Ilves, V. P. Pilipenko "Software-hardware and technological facilities of a remote information controller point" (Institute of Cybernetics, Ukrainian Academy of Sciences, Kiev), A. V. Sukhobokov "External task planner: Status and development prospects" (Tashkent Polytechnical Institute, Tashkent), Ye. S. Trostin "Order, accounting and monitoring of terminal time in OS RV-type system" (Chelyabinsk), A. F. Snizhko "Modeling of data processing management at the main computer center, republic automated computer center, Ukrainian SSSR" (Main Scientific Research Computer Center, Ukrainian State Planning Commission), and V. A. Korolenko "Automation of planning at the computer center" (Minsk).

An interesting report was presented by V. B. Dralyuk (Peddagagix Institute, Sverdlovsk), in which a classification was presented of the interfaces of various components in the computer process with a multiuser operating system, unified principles were formulated for implementing such interfaces including a detailed analysis and development as applicable to the YeS computer operating system. Based on these principles, a number of methods were discussed for dynamic support of the interface of programs with the operating system, allowing dynamic adjustment to specific operating system configurations[3].

During the discussion of these reports, it was noted that as computer systems become more complex (multimachine systems, computer networks), problems of managing the data processing technology are moving once more into the focus of many developments. The role of software and organizational facilities to track the computing process in increasing the effectiveness of computer use has once again, contrary to predictions (resulting from the massive use of personal computers) not decreased, but rather increased. Furthermore, the organizational level of a specific information technology may represent a general quality characteristic of the degree of its sophistication. The increasing influence of programming

[3]Dralyuk, V. B. "Multitasking Methods Of Servicing Subscribers Of The 'Argus' Dialogue System," USiM, 1987, NO. 2, pp. 28-32.

technology on computer center or computer network operating technology was also noted, and the lack of comparative quantitative characteristics of different technologies for preparation of program products was mentioned.

The speakers mentioned again the difficulties encountered in organizing computer center operations for large scientific research institutes. In the organizational aspect, such a computer center is a supplementary subdivision, and thus not always a "favorite," in spite of the fact that the intellectual potential of the computer center employees is high, while their work in terms of its level of complexity, intellectual and physical effort is equal to that of the purely scientific subdivisions. "External" problems of the organization of computer center activity make the internal problems more difficult: An incomplete approach to utilization of hardware, recording of failures, accounting for and planning of work with a shortage of almost all resources; "friendly" software (widely used languages and compilers still at the level of the 60s, diagnostics oriented toward the programming professional). The solution of these problems is seen in the use of operating systems which, both when operating in dialogue mode and with text, can service all computer center users within the interconnected and separated technological lines (i.e., specific information technologies).

In recent years, there has been a lag in the creation of facilities for automating the planning and management of the basic productive activity of various types of computer centers. Unfortunately, until recent times problems related to the computation of the necessary productivity of computer installations, selection of types and configurations of computers, composition of annual plans for provision of machine resources to users, monthly and weekly plan schedules for allocation of machine time, accounting for the use of machine resources and user billing were insufficiently supported by scientific-methodological developments and frequently were solved in a disconnected manner, with no unified methodological basis. Many functions performed by computer center personnel involved in planning the use of machine time and organizing the computing process are not automated.

Many tasks have assigned times for arrival of information and output of results to users. The annual planning cycle is performed over 13 months. Each year the developers of the annual plan include new tasks, and users make changes to previous tasks. Changes to tasks require restructuring of the technology used to perform them; configuration of required programs from individual modules. The changes made are frequently not documents, due to time shortages.

Many task algorithms require the direct participation of specialists comparable in their qualifications to system developers, with great knowledge in the area of the processes they control, organization of application programs and data, and the technology of task performance. The sequence of program start up and communications between programs are described in the huge volume of documentation at the center, too great to be comprehended. Even today, computer centers must have operating subdivisions specializing in the performance of the tasks of each subsystem individually.

Position is made more difficult by the fact that the subsystems in use do not have a common database, monitor programs to control dialogue, all of which were developed using different programming languages and in many cases function under different operating systems. Situations frequently arise in which there is a shortage of machine time. It is not always clear whether available machine resources are efficiently used, or whether the personnel are at fault in the situation.

As computer centers are converted to self financing, it will be impossible to have a reserve of computer power, since, first of all, this would not provide the standard loading on the computers and, secondly, the economic effectiveness characteristics of the subsystem would drop, and furthermore the cost of information–computer services would increase, arousing protests from subsystem users. Therefore, the accuracy of computation of the efficiency of automated management systems must be improved, along with the existing system for organizing the computing process.

Problems of predicting the time required to perform automated management system tasks in multiprogramming mode need more work, particularly tasks which have not yet been experimentally operated. Furthermore, the time required to perform a task in multiprogramming mode depends on the mixture of tasks being simultaneously performed, which cannot be predicted in advance. The technology of task performance in some subsystems continues to be nonautomated.

During the discussion of problems relating to improvement of computer center operating technology, the two most pressing tasks for improvement of information technology management systems were revealed:

— Improving the level of organization of data processing technology, including selection of criteria for evaluating optimal organizational conditions of information technology (in each specific form) and selection of a characteristic to evaluate its organizational level;

— selection of effective versions for utilization of elements of the new information technology (processing, transmitting, input and display facilities), including organization of the computing process. The criterion here is comparability of results and the costs of the new equipment.

Analysis of the capabilities for using new information technology must consider the functionality of a specific device and practical suitability for use (facility of use, ease of loading, ease of repair, etc.).

It was noted that information technology management systems should distinguish among technological, economic and organizational management. The criteria of economic management are closely related to costs, and can therefore be considered management of the distribution, transfer and consumption of costs (characterizing the effectiveness of the entire technological system). Organizational management, like economic management, is superimposed over technological management, i.e., creates, organizes and improves it.

Improvement of the organizational and economic levels as applicable to information technology occurs in the direction of so-called external management[4] of multimachine systems and computer networks, in contrast to internal management, performed by the computer operating system facilities.

[4]Danshin, B. N. "Problems Of Creating And Developing Regional Computer Center Supervisory Services." USiM, 1987, No. 1, pp. 14–17.

In the discussion of the reports it was noted that the creation of new software management resources for the computing process must involve complete automation of individual technological data processing elements and subsequent combination of these elements into a single, fully automated system, satisfying the functional, economic, organization and technological requirements. This also creates the necessary conditions for introduction of the most efficient organizational forms of utilization of computer equipment (personalization of computations, shared use of expensive hardware, etc.).

Improvement of information technology management systems is not a single action, but rather a complex, cyclically developing process, in which practical activity must be alternated with theoretical summarization of results obtained in order to determine the directions for future studies.

The fruitful work of the seminar was facilitated by an exhibition of promising new developments in the area of computing process management, excursions to computer centers, creative contacts between developers and users. It was gratifying that during the course of the seminar, developments were exchanged on the basis of scientific and technical cooperation, significantly speeding up the introduction of these developments to practice, allowing more rapid determination of their strong and weak points.

The next session of the seminar is planned for September of 1988.

Development of Dialogue Task-Oriented Systems

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian No 3, May-Jun 1988, pp. 122-123

[Article by N. P. Trifonov and V. Sh. Kaufman]

[Text] One distinguishing property of dialogue task- oriented systems, doubtless of importance for the future, is the presence of a certain metaalgorithm modeling a familiar technological task performance chain from the professional experience of the user. It is not important to the user how the system is implemented — on a personal computer or a terminal attached to a powerful computer system.

Of course, the authors of the monograph here reviewed[1] recognized the difficulties in achieving this goal, among which is the development of the concept of semantic (or conceptual) programming and the related lack of unified terminology, the acceleration of the rate of modernization of computer equipment and the resultant obsolescence of familiar ways of understanding the organization of performance of tasks on computers. However, the coming advent of fifth-generation software justifies the goal of the authors — to cast off the accumulated system programming baggage.

This monograph can be arbitrarily divided into two parts. In the first four chapters, the experience gained in developing software is systematized, while chapters 5 and 6 are dedicated to the authors' own development — an instrumented large-module programming system called DISUPPP.

Chapter one presents an overall view of the subject studies — task-oriented systems. A general description of task-oriented languages is presented and trends in their development are described in addition to trends in the development of task-oriented systems. The problem of generating a task-oriented system is studied and the task of generating dialogue task-oriented systems is outlined within the broader problem, the complexity of which has still not been fully recognized by most developers.

Our concept of this problem has seen a shift from the technology of programming itself (with its present model of the "limited" lifecycle of a program product) to large-module, prefabricated planning of task-oriented systems by aggregation of knowledge and subsequent modeling of subject areas. The authors state that task-oriented systems will be constructed on the basis of production systems or logical (deductive) conclusion processors, so that the major task in creating a specific task-oriented system will be reduced to development of a task-oriented subject-area model and establishment of its correctness (completeness and consistency).

Chapter two discusses methods of increasing the reliability of task-oriented systems. In addition to traditional methods of testing and verification of programs, approaches for establishing the correctness of a subject-area model in various task-oriented systems are studied. Experience has shown that these approaches combine mechanisms of testing and

[1]**Perevozchikova, O. L., Yushchenko, Ye. L.** Sistemy dialogovogo resheniya zadach na EVM (Dialogue Problem-Solving Computer Systems). Kiev, Nauk. dumka Press, 1986, 264 pp.

verification of programs, although approaches to task-oriented system quality evaluation based on proof of subject-area model correctness have not yet been produced. In the opinion of the authors, it has now "become clear that we need to verify not programs, but rather subject-area models or knowledge bases, in other words, the dynamic software product. For programming in general, the most direct and rapid dialectic relationship between theory and practice in all areas of human activity is now characteristic, and it would be quite incorrect to call individual trends in the development of the theory impractical."

Chapter three of the monograph presents an evaluation of task-oriented system planning technologies. The authors have borrowed their definition of workability from the polytechnical dictionary edited by I. I. Artobolevskiy, "... agreement of a product with the requirements of an economic technology for its manufacture," and use it to evaluate the workability of a software product, as achieved in some domestic instrumented technological complexes such as PROYeKT, RTK, TKP, APROP, ARIUS, ALTOP, MARS, YaUZA-6, TEMP and others. The authors note that in order to achieve the desired level of workability "... the distance from the separateness of present methods used to formalize individual semantic aspects of the language and the community of the dynamic model of the lifecycle software must be overcome to construct an abstract software model considering the multiplicity of levels in the processes of planning and use of software.

The product of some module programming systems, among which the authors include PRIZ, DILOS, ARPOS, VOPROS-OTVET-2, Δ-STATE, VIKAR, VILNYuS and DISUPPP, in quite workable. These products are task-oriented systems with various types of subject-area models, differing both in their semantic power and in the methods of planning and organization of computation.

Chapter four of the monograph investigates approaches to standardization of programming, accumulated in some individual areas: Programming languages, super linguistic "processor environment (i.e., methods of system implementation), database management, remote processing in computer networks. The authors note correctly that the role of standardization in programming differs essentially from "... the role and position of standardization in other areas of human activity, such as in machine building. In programming, standardization serves the theoretical synthesis of scientific knowledge — a process directed toward unification and interconnection of previously separate disciplines or accumulated data, composing factual materials within the framework of a certain scientific discipline ... As a result, the contradiction between the separate, separated facts, and the common, new principles and laws, synthetically combining these facts, is resolved."

The results of study and systematization of extensive factual material accumulated in the process of unification and standardization in programming are used by the authors to perform a comparative analysis of OS YeS monitors. Among the more than 30 available, 17 remote monitors were selected, the common portrait of which rather completely reflects the entire spectrum of problems involved in the maintenance and debugging of software in the OS YeS environment. Among the remote monitors discussed are both the standard remote monitors SRV, DISP, DUVZ, as well as individually developed systems PRIMUS, JACK, FOKUS, the CAMA-dialog system, VEKTOR, DESNA, etc.

The remote monitors were compared as to their functional characteristics and computer resource management dynamic factors. These factors result from the architectural specifics of the monitors. Tables of functional characteristics of the monitors are presented, equivalent to their certification for satisfaction of the needs of three groups of users: Novices, application and system programmers. Users are divided into these groups not based on their knowledge level, but rather based on the roles which they perform at any specific moment of interaction with the remote monitors.

Thus, the certification of remote monitors is not performed in terms of subjective OS YeS software debugging and maintenance categories, but rather on the basis of objective numerical monitor metrics. These metrics generally reflect the workability of the remote monitor products, which is significantly inferior to the instrumental-technological systems.

Chapter five analyzes dialogue path systems — products of the DISUPPP instrumental system, generated by assigning two-level subject area graphic models. The specifics of the path systems include the support of single-step dialogue computations based on the subject-area model, during which the user can select alternative decisions (paths) for tasks, restart and interrupt computation, repeat path segments, generating information on the status of computation and modifying initial and intermediate data. A single information base is used to support multistep computation, reflecting the sequence of system path states.

Path systems embody the most progressive methods of problem solving: Dynamic planning using two-level graphic subject-area models, organization of iterational multistep computations based on coprograms, management and segmentation of a single information base, constant naming of segments and associative data retrieval based upon them.

Chapter six of the monograph describes the process of generating a path system in DISUPPP: From description of the subject-area model in the specification language to modularization of path system components and establishment of subject-area model correctness (including correctness of iterational computations).

The complexity and size of DISUPPP can be described by an anecdote concerning just one stage of production of this product. To determine the most effective rule for modularization during configuration of modules into path systems, two generation systems were studied: With mixed and with external modularization. It was found that with external modularization, the overhead of the use of various OS YeS processing programs (compilers, linkage editors, etc.) did not influence the relationship between subject-area model complexity and machine time expenditure for generation of path systems.

The great workability of path system is explained by the successful combination of advanced subject-area model formalization methods and the huge potential of standard computation support facilities in modern operating environments, clearly underestimated by many developers.

This monograph summarizes the tremendous experience in development of task-oriented systems. Many dozens of systems are named, methods of their implementation are described and systematized. One could debate the completeness of the systematization, but the contribution of the authors to recognition of the most important system programming tasks related to implementation of task-oriented systems and transformation of traditional statements of these problems in a new light is beyond doubt.

The monograph will doubtless be useful to a broad range of computer software developers.

Problems of Development of Information Systems

81440014 Kiev UPRAVLYAYUSHCHIYE SISTEMY I MASHINY in Russian No 3, May-Jun 1988, pp. 123- 125

[Article by M. R. Kogalovskiy]

[Text] For more than two decades, database technology has occupied a firm position as the basis of creation of modern effective information systems. In the initial stage of its development, in the 1960's, new approaches to data management were simply formulated, the first attempts were made to create commercial DBMS. In the 1970's, thanks to the active research in the area of data models and system architectures, database theory began intensive development, experimental work was undertaken in the area of database planning methodology, distributed systems and database machines. The present decade can be considered the age of formation of commercial technology for information system development, integration of research and development related to databases, programming languages and artificial intelligence, with the development of smart database systems and the creation of systems of a new type — expert databases. Database technology has grown into an independent and rapidly developing branch of information science.

In recent years, many scientific and technical works have been published by both Soviet and foreign authors on this problem. However, the bookshelf of the specialist is still not too rich in methodologic publications containing serious analysis of the status and prospects for development of this area, systematizing the problem of planning, development, introduction and utilization of large systems of this class and, in particular, the methodologic aspects of the industrial technology for development of various systems of this type.

In this respect, the publication of the new monograph[1] of G. I. Fursin, a specialist known primarily for his work in the area of programming tools and industrialization of the development of large database systems for automated management systems, the preparation of industry-wide documents regulating this type of activity, is doubtless and event of great interest.

This small book (about 10 printed sheets) with its broad name was greeted at first with some skepticism. However, the very first pages the reader becomes convenience of the seriousness of the authors intentions and the firm foundation of his arguments. The range of problems analyzed in this book is very broad — the status and prospects of development of information systems, problems of improving their intelligence, optimization of planning, industrial development technology.

Chapter one starts (in accordance with the rules of good taste) with an introduction of the basic concepts which will be used in the book and their definitions, presenting the concepts of multilevel information system architecture. This material, vital to the subsequent presentation, is presented quite clearly and accurately. We note only the absence of references to the basic report ANSI/X3 SPARC (1975), in which the architectural approach here analyzed was first formulated.

[1]**Fursin, G. I.** Teoriya i praktika sozdaniya bankov dannykh (Theory and Practice of Database Creation). Kiev, Vishch. shk. Press, 1987, 192 pp.

The material of chapter one is basically review material. Its purpose is to provide the reader with a general orientation in the subject area to be analyzed. Thus, a classification of information systems from various standpoints is presented, helping to understand the reasons for the variety of their functions and engineering characteristics, the spectrum of possible areas of application. A separate section is dedicated to problems of increasing the intellectual level of information systems, their relationship with expert systems and knowledge base systems. Finally, important problems related to the development of information systems based on database technology, called databanks, are briefly analyzed. Since the material is presented from the architectural standpoint, the author intentionally refers t o the subheadings as "Three levels — three groups of problems."

Chapter two, a brief chapter, is dedicated to analyzing the basic aspects of conceptual modeling and requirements placed on tools for its implementation. Conceptual modeling is studied not only as an important trend in research and development, but also as a specific form of activity in the process of creating an information system. Concepts of conceptual modeling, as we know, have been formulated as a result of computation in information system architecture at the so-called conceptual level, first suggested in the report ANSI/X3/SPARC mentioned above. In recent years, conceptual modeling has been actively developed due to the recognition of the commonalty of its role in database technology, programming languages and artificial intelligence.

This chapter has a special place in this book. Based on recognition theory and the dialectic method, the author has found a conveniencing and natural interpretation of approaches and principles which make up the basis of conceptual modeling. In our opinion, this discovery by the author is of fundamental significance.

Chapter three, the longest in the book, presents a detailed discussion of a semantic data model developed by the author and his colleagues, intended for conceptual modeling purposes. In spite of the somewhat old-fashioned name — KOMOD, the model is quite modern as to the ideas upon which it is based and well developed as to its functional capabilities. Unfortunately, the most positive attitude toward this book cannot overcome the impression of dissatisfaction with the last few sections of this chapter. The few plan solutions related to experimental implementation of this model presented in these sections are somewhat unconvincing and architecturally illogical, and there are terminological errors.

The material in **chapter four** is quite interesting and presents fresh ideas. The authors demonstrates here just how the methods of multicriterion optimization developed at the Institute of Cybernetics, Ukrainian Academy of Sciences, can be applied to database planning. In the mathematical model suggested, he takes as his basis certain ideas related to the analytic estimates of database characteristics presented in the well-known monograph of T. Tiori and D. Fray, Proyektirovaniye struktur baz dannykh (Planning of Database Structures). Moscow, Mir Press, 1985. The approach developed by the author is implemented in his plan for ALTERNATIVA, the main concepts of which are also discussed in this chapter.

The elements of the novelty here include both the initial statement of the problem and the attempt to use a known mathematical apparatus within the framework of a new sphere of application.

Chapter five, the last chapter in the book, dedicated to problems of industrialization of information system planning, is very important. On the example of a real database, included in a construction automated management system, the basic methodological developments of large information systems are studied, based on the conceptual modeling facilities and multicriterion optimization techniques from Chapter four. The characteristic features of large databases used within enterprise automated management systems are discussed, as are the functional capabilities of the software systems used to create and operate such systems, the technological plans for their development and assurance of functioning by the use of these tools.

The book also contains appendices presenting the strict specifications of the syntax of the linguistic facilities used in the KOMOD data model and sample plan documents for the databases of Chapter five, as well as a conclusion section which presents a brief review of the most important trends in database research and development.

In spite of the brevity of the analysis of a number of problems, due to the limited length of the book, a few terminological errors and omissions in the bibliography, this monograph is a definite success for the author.

The work reflects the latest achievements in database theory, summarizing the practical experience of their development. A significant portion of the work consists of a discussion of the results of the authors own research, containing a number of original ideas and nontraditional approaches.

In our opinion, the book is doubtless useful, both for beginners and for experienced specialists.

- END -

10