Volume 13 Number 3 July 1989
YU ISSN 0350 – 5596

Informatica

A Journal of Computing and Informatics

The Slovene Society INFORMATIKA
Ljubljana

Informatica

A Journal of Computing and Informatics

Subscription Information

Informatica (YU ISSN 0350 – 5596) is published four times a year in Winter, Spring, Summer and Autumn (4 issues).

The subscription price for 1989 (Volume 13) is US$ 30 for companies and US$ 15 for individuals.

Claims for missing issues will be honoured free of charge within six months after the publication date of the issue.

Printed by Tiskarna Kresija, Ljubljana.

Informacija za naročnike

Informatica (YU ISSN 0350 – 5596) izide štirikrat na leto, in sicer v začetku januarja, aprila, julija in oktobra.

Letna naročnina v letu 1989 (letnik 13) znaša za podjetja 48000 din, za zasebne naročnike 12000 din, za študente 4000 din; posamezna številka 16000 din.

Številka žiro računa: 50101 – 678 – 51841.

Zahteva za izgubljeno številko časopisa se upošteva v roku šestih mesecev od izida in je brezplačna.

Tisk: Tiskarna Kresija, Ljubljana.

Na podlagi mnenja Republiškega komiteja za informiranje št. 23 – 85, z dne 29. 1. 1986, je časopis **Informatica** oproščen temeljnega davka od prometa proizvodov.

Pri financiranju časopisa Informatica *sodeluje Raziskovalna skupnost Slovenije.*

Volume 13 Number 3 July 1989
YU ISSN 0350 – 5596

Informatica

A Journal of Computing and Informatics

The Slovene Society INFORMATIKA
Ljubljana

Letnik 13 Številka 3 Julij 1989 YU ISSN 0350 – 5596

Informatica

Časopis za računalništvo in informatiko

V S E B I N A

DESIGN AND TESTING OF HOMOGENOUS SINGLE BUS TIGHTLY COUPLED MULTIPROCESSOR SYSTEM FOR REAL TIME SIMULATION

INFORMATICA 3/89

Keywords: multiprocessor system, real time simulation, single bus, parallelization of mathematical models

Krešimir Ćosić, Ivan Miler i Igor Rašeta
VVTŠ KoV JNA Zagreb

ABSTRACT - The real time simulation for hardware or man in the loop testing presents the cost effective way for design, development, modification and testing of a complex and sophisticated weapons and industrial systems. This simulation technology is a constant challenges for most powerful computer systems. Therefore one short chronological review of computer architecture for time critical real time simulation is given. For such kind of application one homogenous single bus tightly coupled multiprocessor system based on 8086/8087 single board computers has been designed. Furthermore, this article presents one concept for parallelization of mathematical models given by ordinary differential equations in real time environment. Simulator design for one spinning missile system, according to the accepted procedure, illustrate the abilities of realized multiprocessor simulator system.

SAŽETAK - Simulacije u realnom vremenu za testiranje realnog hardvera ili operatora u zatvorenoj petlji, predstavljaju efikasan put za projektiranje, razvoj, modifikaciju i testiranje kompleksnih sofisticiranih vojnih i industrijskih sistema. Ovakva simulaciona tehnologija predstavlja stalan izazov za najsnažnije računarske sisteme. Iz tog razloga dan je jedan kratak kronološki pregled računarskih arhitektura za vremenski kritične simulacije u realnom vremenu. Za takvu vrstu primjena, u okviru ovog rada, realiziran je jedan homogeni, čvrsto spregnuti multiprocesorski sistem s jednom sabirnicom i nizom procesorskih ploča baziranih na procesorima 8086/8087. Pored toga, izveden je i prikazan jedan koncept za paralelizaciju matematičkih modela zadanih običnim diferencijalnim jednadžbama. Projektiranjem simulatora, prema usvojenoj proceduri za jednu rotirajuću raketu, prikazane su mogućnosti realiziranog multiprocesorskog simulatora.

1. INTRODUCTION

The increasing complexity and sophistication of modern process control and weapon systems has established a category of real-time simulation which uses hardware components integrated in the process of simulation. This kind of simulation, so-called hardware-in-the-loop (HIL) simulation technology, has proved to be a very cost effective method in design, development, modification, and testing of complex weapons and industrial systems [1,2,3,4]. In HIL simulations, for example, adequate computer equipment can be used to simulate the aerodynamics and flight equations of the missile, while the real hardware subsystem such as RF sensors, IR sensors, fin actuators, autopilots, guidance and homing on board computers can be embedded and used for design and testing of a closed-loop system. In this case, HIL simulation provides a reproduction of what the real hardware subsystem (missile seeker) really processes in real environment, on the basis of simulation of the missile aerodynamics, flight equations and targets movement simulated by suitable pseudo-target generator. In a this way it is possible to perform nondestructive testing, verification and validation of actual missile or process control subsystem in near realistic environments. This preflight check and similar industrial testing in the early phase of design and development provides effective way to analyze the overall performance capability of the closed loop system and to predict a performance at minimal cost.

2. COMPUTER ARCHITECTURE FOR TIME CRITICAL REAL TIME SIMULATIONS

Hardware-in-the-loop and man-in-the-loop simulations, which require time critical real-time simulations have proved to be constant challenges for the most powerful computer systems. Demands for more and more fidelity and accurate simulation of dynamic system characterized by ordinary differential

equations, significantly increase demands for additional speed and power from simulation computers. These demands have increased at the rate at least as fast as the rate of development and improvement in computer technology. Thus, today, available simulation capabilities have the same relationship to the requirements as 10 years ago [3]. Furthermore the speed requirements of the more challenging simulation applications, very frequently exceed the capabilities of even the most powerful mainframe computers. During the 1960's hardware-in-the-loop simulation of time critical processes depended on analog computers, such as EAI 231, EAI 781. In analog computers parallel operations of many computing elements provide very high speed of processing which is the most significant for time critical real-time simulation. Programming of these processors was a manual process using the patch boards and fixed point scaled equations. In early 1970's hardware-in-the-loop simulations has predominantly shifted to hybrid computations i.e. combination of analog and digital hardware, such as EAI PACER 100, to provide the required computational capabilities. In the middle of 1970's with the advent of fast mainframe digital computers the emphasis in application is based exclusively on digital hardware. But at that time the speed requirements of more challenging simulations frequently exceeded the capabilities of even the most powerful mainframe computer such as IBM 360, CDC 7600, Univac 1108 and so on. Today, currently available mainframe supercomputers such as CRAY-1, CRAY X-MP-1, CRAY X-MP-2, IBM 3090/VF-200, NEC SX-1E, NEC SX-2, CDC CYBER 205, Amdahl 1200, Hitachi S-810/20 and so on, in majority of cases provide necessary computational power, but very often doesn't provide the cost effective approaches for such applications. Therefore in the late 1970's the trend in architecture of a digital computer system for time critical real-time simulations was toward the more specialized architectures. The first of these devices was peripheral array processor AD-10 (1979) manufactured by Applied Dynamics Inc. It was simulation-oriented peripheral processor intended primarily for the simulation of systems of ordinary differential equations. Performances of this system are given in Table 1. They are related to estimation of computer power necessary for development of helicopter simulator.

Impressive speed of the machine when applied to ordinary differential equations results from its advanced technology which provides very high processing speeds and by the extensive pipelining and parallelling and from specialized computing and memory units suitable for solving nonlinear ODE-s [7]. New version of this system AD-100, which is characterized by significant improvement in performance (Table 2) has appeared on the market in 1984.

TABLE 1

HELICOPTER SIMULATORS [5]

computer	and achieved frame times		
CYBER 175	45 ms	FPS AP-120B	4.5 ms
CDC 7600	15 ms	AD-10	0.8 ms

TABLE 2

MODEL OF A WHIRLING FLEXIBLE BEAM [6]

computer	and AD-100	advantage	
ADI AD-100	1.00	IBM 3033	7.45
CRAY 1-S	3.35	FPS 164	17.95
IBM 3081	5.40	HEP H-1000	36.65

But very often hardware-in-the-loop real-time simulation requires simulator systems that are more cost effective and portable. The first cost effective way for attaining analog computer speed leads to parallel operation of multiple digital microprocessors. However, the price/performance ratio of multimicroprocessor system was very attractive and therefore a number of attempts to interconnect relatively inexpensive general-purpose microcomputers have been made over the past years for designing a complex simulator systems. With the increasing availability of very fast and very economical single board computers, it becomes feasible to design the construction of network of microprocessors which will form special-purpose simulator. This approach was very attractive and more favorable in speed/cost ratios in relation to other solution. Therefore a number of microprocessor networks were developed for real-time simulation throughout 1980. The real-time multiprocessor simulator (RTMPS) project at the NASA Lewis Research Center for the simulation of jet engines (1984) was one of the first and most significant [8]. The recent introduction of powerful multiprocessor systems by a large number of vendors (1985-1987), such as Ametek Computer Research Division, Alliant, BBN Advanced Computers, Elxsi, Encore Computer, Flexible Computer, Intel Scientific Computers, Ncube, Thinking Machines and so on, has increased the interest of engineers and scientists in this approach to high speed real-time scientific computations. But it is necessary to maintain that this approach is not cost effective and quite attractive for the majority of customers and simulator vendors. Today, very cost effective approach to parallel digital real-time simulation is based also on the network of transputers [9,10]. The T800 Transputer contains a 10 MIPS 32-bit processor, on chip RAM, timer and I/O interfaces which are based on serial communication channels. T800 have four links per chip and they use a clock rate of 20 MHz on the serial link, so that the communication channel between two Transputers requires only the connection of two wires for the link. Two T800 Transputers with floating point hardware with a speed of 1-2 MFLOPS, in the simulation of 2-nd order system by the RK 4 integrators is

approximately twelve times faster than the Intel 80386/80287 and six times faster than the Motorola 68020/68881 [10]. Further evolution of transputer networks by Inmos Inc. and Micro Way provides abilities to design and build arbitrarily large parallel processing machines. The 32-transputer array has been used in simulator design for modelling the flow through a jet engine's turbine-blade cascade by the Rolls-Royce at Derby [11]. This model has required ten minutes to run on a 32-transputer array and two minutes on a Cray XMP-48. Since the Cray cost over 125 times as much as 32-transputer array, the price performance ratio is 25:1 in favor of the transputer network.

But parallel operation and parallelism doesn't guarantee performance, and may, in fact, limit it. Example for this is successful replacement of 64-processor system Illiac IV with higher performance serial processor Cray I. The number of processors, interprocessor communications, memory organization and numerous other factors interact to limit or to enhance processor performance. The effective utilization of a network of processing elements or microcomputers poses difficult scheduling and allocation problems. This means that the major difficulty in using parallel processor is the effective software support, so that the total performance improvement in relation to the serial processing is dependent in the same time on the numerical procedures which are used i.e. techniques of discretization, techniques for decomposition and then, on the power of hardware for simulation.

3. ARCHITECTURE OF THE REAL-TIME MPS-AMS MULTIPROCESSOR SIMULATOR

The real-time Multiprocessor Simulator - MPS organized on a shared single bus -AMS, shared dual-port memories i.e. on tightly coupled multiprocessor topologies [12] is shown in Figure 1. Modular design of this system provides a number of benefits which are related with its flexibility in modification, reconfigurration and maintenance [13]. Four Siemens 8086/87 based single board computers (SBC) AMS-M8-A8 are used to realize simulator hardware on principles of master/slave relationship. SBC's boards are connected through the 16-bits data bus AMS-M (European realization of the IEEE 796 Multibus I) which supports the real-time processing features. Access to the analog I/O world has been provided by the 16/32 channels analog to digital input board 12-bits AMS-230-A1, and by the four digital to analog channels 12 bits, realized on AMS-M596 standard bus interface board. Such system is characterized by low functional complexity, physical compactness, and relatively low-cost. Communication among the processors is performed via message passing in "mailboxes" that reside in distributed dual port memory. Access to this memory occurs via a single time shared bus.

In the phase of the configuration of real-time multiprocessor simulator, development of the simulation real-time software requires selection of the modules, their editing, compiling and linking on the host PC XT system. The following step in the procedure is related to the assembling modules according to the block diagram of simulated process or system and their testing, evaluation and validation. After that the tested modules are downloaded onto the master processor board and finally, created modules are mapped from the master boards to the slaves boards of MPS according to the accepted decomposition schemes [14].

Furthermore the host system provides the overall control of the simulation activities through the suitable graphic language. With additional multiuser microprocessor development system Tektronix 8560 i.e. through different integration station Tektronix 8540, this system enables efficient development of custom design hardware and software modules [15]. Two processors, host and master, communicate using interrupt system via a PC bus window, through the high speed SMP - PC interbus SMP-E570-A1. This technique is selected as the fastest available communication between the two processors, which allows one system to access the address on a companion system's bus as through the address on its own bus. To prevent conflicts in sending and receiving data between PC-XT and AMS system through PC memory, the synchronization mechanism uses the flag test-and-set procedure (semaphore) [16].

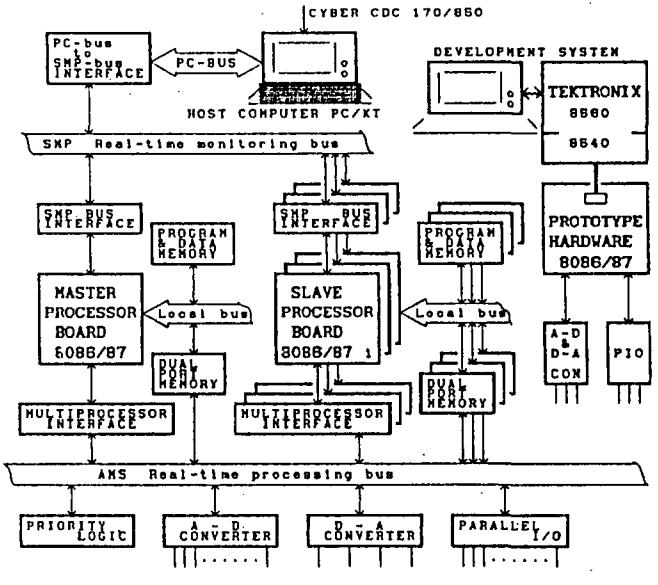


Figure 1. Multiprocessor architecture for Hardware-in-the-loop testing

AMS-M bus protocol defines master-slave communication and multiprocessor arbitration which is strongly problem dependent. A real-time monitoring SMP-M bus is 8/16 bits data bus architecture with its own bus interface on SBC board but without multiprocessor arbitration. The local bus on the AMS

boards connects the processors to all on board input/output devices (24 lines PIO 8255, RS 232 SIO 8251), local memory (EPROM 2764, SRAM 6116) and communication memory (dual port memory SRAM 6116), as shown in Figure 2. This bus permits independent execution of onboard activities.

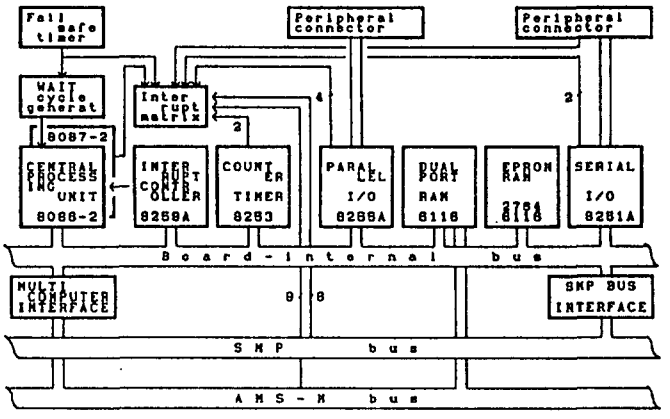


Figure 2. Single board computer AMS-M6-A8

A backplane provides the physical connections of AMS-M and SMP-M bus signals and priority resolver lines to set the priorities. Each board has a fixed priority which can be changed through jumpers on the backplane.

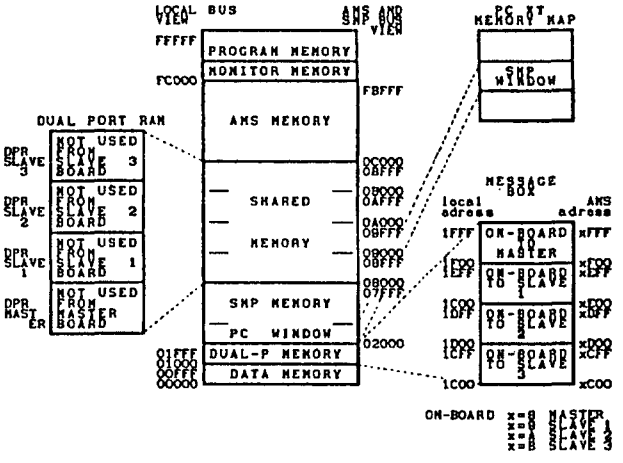


Figure 3. Multiprocessor memory organization

In this tightly coupled multiprocessor configuration, the processors communicate over the parallel bus, AMS-M, through a common i.e. shared memory. Generally speaking the common memory can be concentrated or distributed. In the accepted configuration, common memory is partitioned on each processor board as dual-port memory (DPM) to reduce bus occupation. Additionally, each processor board has a local memory which is especially interesting in loosely coupled decomposition algorithm which frequently use only local memory and seldom common memory. Furthermore, at the same time, this type of memory organization allows parallel access to shared memory without using the AMS-M real-time bus through the local bus. In addressing DPM all read accesses are local and do not use common bus. All write accesses use two different addresses. Onboard addresses are used to address its local memory and local dual-port RAM. Addressing DPM on the other boards are provided through AMS bus controller in memory space 0x000-0xFFF depending on the selected boards. The memory organization of this system is shown in Figure 3. To access the shared memory, it is necessary to gain the AMS-M bus, which then is locked-on through standard protocol.

The most important aspect of the bus interconnection topology used in this MPS is the bus arbitration technique. The priority level is determined by user through wrapping technic on backplane according to the Figure 4.

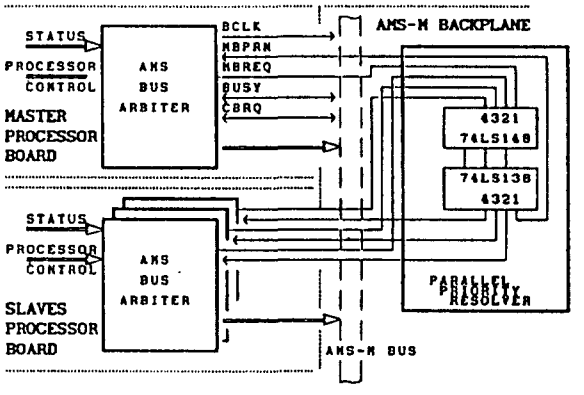


Figure 4. AMS-M bus arbitration

Each SBC has two arbitration lines, bus request (BREQ) and bus priority in (BRPN), which are used to gain access to the AMS bus. BREQ line comes from a SBC to priority resolver and indicates a request for control of the AMS bus. BPRN signal comes from priority resolver to a SBC and indicates that the processor may go ahead and use the bus since there is no other higher priority request for the AMS bus.

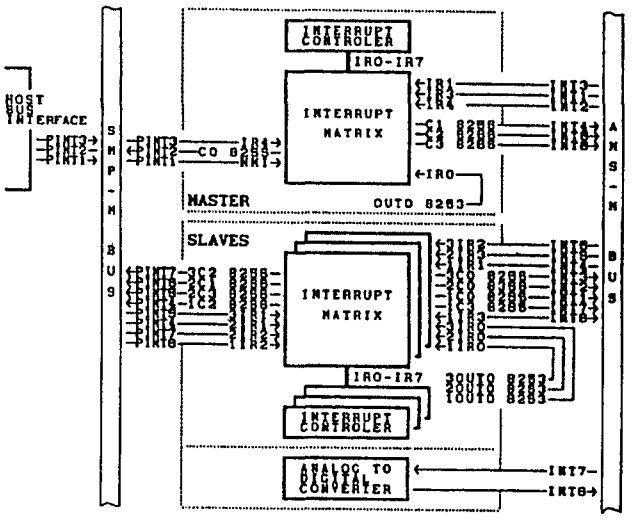


Figure 5. Interrupt system of MPS-AMS

To optimize the communications efficiency and to provide real-time processing of the real-time clock request, MPS system is predominately interrupt driven. The interprocessor communication begins by passing an interrupt request signal from one processor to another. The priority assignment in the interrupt system is problem dependent and is shown in Figure 5. for one typical closed loop guided missile system.

The interconnection strategy is adapted to this structure, so that request for each processor is wrapped to the corresponding interrupt level.

AMS system bus uses the non-bus-vectored mode for interrupts. When an interrupt request line is activated the interrupt controller generates an interrupt vector address and transfers it to the processor over the local bus [17].

4. ONE CONCEPT FOR PARALLELIZATION OF MATHEMATICAL MODELS GIVEN BY ODEs IN DIGITAL REAL TIME SIMULATION

4.0. Real time simulation in multiprocessor environment

The abilities of parallel real time simulation predominantly depend on the performance and architecture of parallel multiprocessor system, types of interaction between parallel processors, on the problem under consideration i.e. its inherent level of parallelism, on numerical methods for numerical integration, strategy of task allocation and finally the cooperation between parallel architecture and parallel discrete time model of a original continuous system. But it is important to note that the main contribution to the improvement of efficiency in multiprocessor real time simulation depends on decomposition of model, on a process of discretization and on mapping of given problem onto parallel multiprocessor architecture. This process usually involves several phases that start with decomposition of mathematical model given by algebraic equations (AEs) and ordinary differential equations (ODEs) or by partial differential equations (PDEs). Since these equations are defined over continuous time domain in the second step some kind of discretization or numerical approximations must be employed in order to enable digital implementation. Finally, it is necessary to perform suitable partitioning of derived discrete time model onto multiprocessor environment. Mapping of decomposed and discretized model onto parallel architectures can be performed in a different way.

In dynamic load balancing, discrete time models are allocated to microprocessors at run time. Discrete models automatically migrate from heavily loaded microprocessor to lightly loaded ones. By attaining a well-balanced load, better processor utilization can be achieved and thus higher performance of complete system.

In a static load balancing method, models are allocated to microprocessors after compile time i.e. before start up run time. Such static techniques require fairly accurate predictions of the resource utilization for each model.

For programs with unpredictable run-time resource utilization, dynamic load balancing is more desirable because it allows the system to continuously adapt to rapidly changing run time conditions. As a popular measure of load balancing it is possible to use CPU time utilization, communication time, the number of concurrent microprocessors in active operation, the number of concurrent models etc..

In real time simulation correct prediction of the computational requirements and resource demands for each software module i.e. for each program must be known in advance to enable real time implementation. This estimation can be derived after the process of discretization. On the basis of this information, algorithm for task allocation provides well load balance and real time execution. Therefore the concept of static load balancing in the real time simulation is a natural one. In simulator design, what is primary interest of our research, the objective function must provide real time simulation of related problem with desired accuracy and with minimal number of microprocessors. The system which enables developments of the simulator in this way can be considered as development system for simulator design and realization. The accepted objective function is natural in designing and realization of digital simulator for operators training or hardware in the loop testing.

Real time simulator described in this article hosted on IBM PC/AT has been realized in order to provide the user with such abilities and furthermore to allow generation of real time machine code for target processor based on Intel microprocessor 8086 and arithmetic coprocessor 8087. Through attached peripheral homogenous tightly coupled multiprocessor system based on single board computers, such simulator system provides user with different experimental abilities in operator training or control system design and testing. The software support of such simulator development system provides the abilities of extensive non real time simulation which leads to such decomposition technique, numerical integration and task allocation strategy which guarantees the minimal number of parallel processing units i.e. minimal hardware complexity necessary for realization of a different kind of simulator system. This hardware complexity strongly depends on desired level of accuracy in process of simulation, so that with increasing level of approximation the number of microprocessor can be significantly increased. The substantial influence on

this facts have choices of decomposition techniques and procedures for numerical integration. Therefore special attention will be dedicated to the relationship between hardware complexity on one side and decomposition techniques, numerical methods for discretizations and algorithm for task allocation on the other side.

Now we can define the procedure for real time simulation in multiprocessor environment with following steps:

Step 1. Structural and dynamics decomposition of system corresponds to physical partitioning of problems into a sequence of modules with lower level of complexity.

Step 2. Tuning between numerical methods and sample rates for discretization and mathematical modules according to its nature; linear or nonlinear, time variant or time invariant, stiff or nonstiff, with or without discontinuities, spectral characteristics of input signals and so on.

Step 3. Task allocation process follows physical arrangement of system and in cooperation with numerical methods for discretization determines desired level of granularity in order to achieve equal load balancing between processors, minimal hardware requirements and real time execution.

Presented procedure is not straightforward, it may be iterative one and all these steps must be taken into account very briefly in order to achieve optimal real time simulation of given problem in accordance with the accepted objective function.

4.1. System decomposition

The first step in the above procedure requires decomposition of a mathematical model of complex dynamic system into a number of hierarchical functional modules or blocks of different complexity. With such a modular or block approach a realistic complex problem can be subdivided into a sequence of smaller modules or blocks. By applying this formalism, complex mathematical models can be easily transformed into one block level distributed structure which enables isolation of standard mathematical models and their further efficient processing. Decomposition scheme, which we shall prefer, is heuristic one and is mainly based on physical partitioning of complex system. Now we shell define the parallelism degree of model on level of block diagram,as the maximum number of functional blocks that can be executed at the same time on different processors if necessary in real time execution. For very fast system i.e. for time critical real time simulation, inherent parallelism degree can be further increased by partitioning from block level to an equation level or even an arithmetic operation level. If it is necessary, it is possible to determine computation of the longest execution time i.e. critical data flow trajectory in running through the block diagram. The efficiency of modular partitioning is problem dependent, but only with such an approach it is possible to perform the optimal adaptation of numerical methods of discretization to each module of distributed system. Such decomposition of a complex system into multiple low level computational modules which enables adaptation of numerical method and period of discretization to each module is most important for digital real time simulation. After structural decomposition which corresponds to physical topology of the system, suitable dynamic decomposition is necessary in order to adapt periods of discretization or integration to each module. The concept of multirate sampling enables different sampling rates in different modules or in different loops, and leads to the significant reduction of computational load i.e. CPU time savings and to improving the numerical conditioning. Such structural and dynamic decomposition detects the level of parallelism which is very often inherent in complex original continuous system. Separation of differential equations according to their type, particularly linear differential equations from nonlinear e.g. nominal trajectory from perturbated, separation according to the spectral characteristic i.e. separation of fast portions from slow ones and separation according to the frequency contents of input signal are substantial for the achievement of high efficient digital real time simulation.

A heuristic method for system decomposition used in this concept is based on the fact that the modules that follow from partitioning of the overall system are very similar to the physical topology of the system. The main advantage of this approach is that the processing blocks are associated with physical sections, and model implementation and verification can be easily done. In addition, variables used in the interprocessor communication have physical meaning, which can aid in the understanding and interpretation of the model. Furthermore by exchanging only the output variables between blocks it is possible to reduce significantly communication requirements between processors.

Because decomposed system is similar to the physical system, this method is problem dependent. However, the advantages of the method far outweigh this disadvantage. The main advantages of this method are:

- Decomposition is easy to make, since it follows the physical arrangement of the system.
- Highly modular approach is very flexible in the case of structural or modules modifications.
- Interprocessor communication requirements are minimized.
- Program design is simplified, which is a direct consequence of modular structure.
- Program coding is straightforward. It is easier

to code the small modules that result from the decomposition than complex ones.

- Checking, testing and debugging is also easier, especially message interchange between microprocessors during interprocessor communications.

This decomposition has been applied on original continuous models and its abilities are determined by the nature of the problem. Modules or blocks that are independent enable simultaneous or concurrent calculation and provide parallel decomposition. Modules or blocks that are sequential lead to the cascade decomposition. This level of parallel or sequential computation is predominately determined by the nature of the problem, but in the following steps it will be shown that this inherent level of parallel or sequential properties can be significantly modified by the choices of numerical methods for discretization or integration.

4.2. Numerical integration

This step transforms original continuous mathematical model into equivalent discrete one, that must guarantee desired level of approximation with minimal arithmetic complexity normalized on some common frame time. By this transformation using different techniques of numerical integration it is possible to additionally increase the degree of model parallelism from the inherent one to some desired level which enables real time simulation of related problems on given multiprocessor configuration. It means that the complete parallelism in discrete time model for implementation depends not only on efficiency of functional decomposition i.e. topology of system, but also on numerical method for discretization. The level of parallelization which will be added or extended with the choices of numerical methods determines the final level of granularity of discrete time model for implementation. The level of granularity in equivalent discrete time model can vary in a range from instructions and statement level to program or a group of programs level. This depends on the complexity of mathematical model and its dynamic characteristics i.e. real time constraints and architecture of multiprocessor system that would be used for implementation. For example, if a model of some problem is represented by a sequence of sequential modules or blocks which require sequential computation, by using any of explicit numerical methods for integration, they can be easily transformed to parallel procedures until the desired level of granularity is reached. Assignment of the suitable integration or discretization method to each module or block, and suitable period of discretization require extensive non real time analysis, checking, testing and verification. In the following part of this article we shall focus our attention on numerical methods that can be considered as suitable for digital real time integration of ordinary differential equations and on dependance analysis between the parallelism of simulation models and numerical methods used for discretization or numerical integration.

For a linear or linearized problem given by

$$\dot{x}(t) = A\,x(t) + B\,u(t) \quad , \quad x(0) = x_0 \tag{1}$$

some modification of standard discretization methods can be considered as optimal one.

Modification of standard step invariant method [20,21] is given by the following equations

$$x^*_{k+1} = \Phi(A,T,W)\,x^*_k + \Gamma(A,B,\lambda,\gamma,T,W)\,u_k \tag{2}$$

where the state vector, and system matrices are determined by

$$\begin{aligned} x^*_k &= W^{-1}x_k \\ \Phi &= W^{-1}e^{AT}W \\ \Gamma &= W^{-1}\lambda[A^{-1}(e^{AT}-I) + \gamma T(e^{AT}-I)]B \end{aligned} \tag{3}$$

State space version of T-integrator is given by [22]

$$x^*_{k+1} = A^*(A,L,\Gamma,T,W)x^*_k + B^*_1(A,L,\Gamma,T,B,W)u_{k+1} + B^*_2(A,L,\Gamma,T,B,W)u_k \tag{4}$$

where is

$$\begin{aligned} x^*_k &= W^{-1}x_k \\ A^* &= [I - LT\Gamma A]^{-1}[I + LT(I-\Gamma)A] \\ B^*_1 &= [I - LT\Gamma A]^{-1}\,LT\Gamma B \\ B^*_2 &= [I - LT\Gamma A]^{-1}\,LT(I-\Gamma)B \end{aligned} \tag{5}$$

For general nonlinear problem given by

$$\dot{x}(t) = f(x(t),t,u(t)) \quad , \quad x(0) = x_0 \tag{6}$$

following numerical integration algorithms can be considered as suitable for real time digital simulation [23];

* Single-pass integration algorithm
  - Euler explicit

$$x_{n+1} = x_n + T\,f(x_n,t_n,u_n) \tag{7}$$

  - AB - 2

$$x_{n+1} = x_n + T/2[3f(x_n,t_n,u_n) - f(x_{n-1},t_{n-1},u_{n-1})] \tag{8}$$

  - AB - 3

$$x_{n+1} = x_n + T/12[23f(x_n,t_n,u_n) - 16f(x_{n-1},t_{n-1},u_{n-1})] + 5f(x_{n-2},t_{n-2},u_{n-2})] \tag{9}$$

* Real time Runge-Kutta version
  - RK - 2

$$\begin{aligned} x^P_{n+1/2} &= x_n + T/2\,f(x_n,t_n,u_n) \\ x_{n+1} &= x_n + T\,f(x^P_{n+1/2},t_{n+1/2},u_{n+1/2}) \end{aligned} \tag{10}$$

  - RK - 3

$$\begin{aligned} x^P_{n+1/3} &= x_n + T/3\,f(x_n,t_n,u_n) \\ x^P_{n+2/3} &= x_n + 2T/3\,f(x^P_{n+1/3},t_{n+1/3},u_{n+1/3}) \\ x_{n+1} &= x_n + T/4[f(x_n,t_n,u_n) + 3f(x^P_{n+2/3},t_{n+2/3},u_{n+2/3})] \end{aligned} \tag{11}$$

* Adams-Moulton serial predictor-corrector
  - AM - 2

$$\begin{aligned} x^P_{n+1} &= x_n + T/2[3f(x_n,t_n,u_n) - f(x_{n-1},t_{n-1},u_{n-1})] \\ x_{n+1} &= x_n + T/2[f(x^P_{n+1},t_{n+1},u_{n+1}) + f(x_n,t_n,u_n)] \end{aligned} \tag{12}$$

-AM - 3

$$x^P_{n+1} = x_n + T/12[23f(x_n, t_n, u_n) - 16f(x_{n-1}, t_{n-1}, u_{n-1}) + 5f(x_{n-2}, t_{n-2}, u_{n-2})] \quad (13)$$

$$x_{n+1} = x_n + T/12[5f(x^P_{n+1}, t_{n+1}, u_{n+1}) + 8f(x_n, t_n, u_n) - f(x_{n-1}, t_{n-1}, u_{n-1})]$$

For complex nonlinear modules or blocks, which unable further physical decomposition and single processor real time implementation, it is necessary to use some technique for equation segmentation or numerical methods for parallel integration. Standard approach for solving such problem [24] requires partitioning the set of n equations and then allocation of a certain number of the n equations to each of the microprocessors to achieve the speed needed for real time simulation. Each microprocessor would be responsible for performing the function evaluations and integrations associated with its assigned equations. Such subset of equations can operate in parallel, only the values which are necessary in the other equations would be transferred periodically between microprocessors. Since the function evaluations i.e. the computation of derivative would be done in parallel, a good deal of time can be reduced in comparison with a single microprocessor implementation. Just how much time is saved depends on;

- How closely coupled is the system of equations;
- How the equations are allocated to the microprocessors;
- What integration algorithm is used for discretization of each subset of equations;
- What types of communication channels and protocols are available between microprocessors.

Alternative solution to the above standard approach is concerned with utilization of parallel predictor-corrector methods. In this case partitioning is performed in the sense of the algorithm i.e. numerical method but not in the sense of the system of equation as in previous approach. One possible parallel predictor-corrector method has been presented by Miranker and Liniger [24] for system $\dot{y} = f(x,y)$ and is given by

$$y^P_{i+1} = y^C_{i-1} + h/3(8f^P_i - 5f^C_{i-1} + 4f^C_{i-2} - f^C_{i-3})$$
$$y^C_i = y^C_{i-1} + h/24(9f^P_i + 19f^C_{i-1} - 5f^C_{i-2} + f^C_{i-3}) \quad (14)$$

For parallel real time simulation parallel block implicit methods can be also very useful. One version of a fourth-order block presented by Shampine and Watts [24] is given by:

- Predictor equations (15)

$$y^P_{i+1} = 1/3(y^C_{i-2} + y^C_{i-1} + y^C_i) + h/6(3f^C_{i-2} - 4f^C_{i-1} + 13f^C_i)$$
$$y^P_{i+2} = 1/3(y^C_{i-2} + y^C_{i-1} + y^C_i) + h/12(29f^C_{i-2} - 72f^C_{i-1} + 79f^C_i)$$

- Corrector equations

$$y^C_{i+1} = y^C_i + h/12(5f^C_i + 8f^P_{i+1} - f^P_{i+2})$$
$$y^C_{i+2} = y^C_i + h/3(f^C_i + 4f^P_{i+1} + f^P_{i+2}) \quad (16)$$

The choice of the optimal discretization method i.e. tuning the method to each separate module requires well understanding of the character and dynamic of each module and well understanding of numerical characteristics of all of the above mentioned numerical methods. But this can be done only on the basis of separation between linear differential equations, and nonlinear, fast portions of the problem from slow portions, time invariant from time variant. It means that the decomposition of problem has great influence on the efficiency of the complete procedure. By such approach we can significantly increase the solution bandwidth of the problem, which is the most important in real time simulation.

4.3. Task allocation

The last step in the presented procedure requires distribution of derived discretized modules among microprocessors in multiprocessor system and can be used in iterative fashion with previous two steps. This process of a task allocation i.e.process of assigning software modules which constitute distributed discretized mathematical model of original continuous system to each processing element, requires an understanding of data or block dependencies that exist among problem variables. There are two different ways in task allocation strategy which depend on applications.

In the first case, architecture of the multiprocessor system is completely defined and determined by the type and number of available processors, their performance and way of their communication. The computing tasks involved in solution of simulation problem, in this case, must be partitioned on the available processors in order to minimize idle time of each processor and to minimize the time lost in the communication of program segments. Furthermore special constraints and limitations on real time processing are not supposed i.e. solution can be generated faster than real time or slower then real time, which depends on the problem and performance of multiprocessor system. In such environment it is necessary to attain such load balancing which leads to a better multiprocessor resource utilization. Well-balanced load provides a higher performance of complete system i.e. minimal total run time. Measures of load balance in this case may include the CPU time utilization for each processor, communication time in relation to the computation time, the number of concurrent processes or modules in simultaneous processing and so on. The combination of these different objectives can be expressed by the fast improvement achieved on multiprocessor system. This improvement can be characterized by factor S, which is ratio between solution times using one processor and N processors[27].

$$S = T_S/T_M \quad (17)$$

where is: T_S - single processor solution time,

T_M - multiprocessor solution time.

The efficiency of complete multiprocessor implementation can be defined by:

$$E = S/N \quad (18)$$

where **N** denotes the number of microprocessors, and **S** is given by (17). Efficiency is expressed as a percentage by multiplying the above expression by 100. The assignment must be made so as to optimize relation (17) and (18), i.e. to enable minimization of execution time.

The second approach to the problem of task allocation is related to design and development of digital real time simulator for operator training or hardware in the loop testing i.e. restricted on real time simulation. Therefore in such applications it may be assumed that the architecture of multiprocessor system is not given in advance and must be determined by this procedure. In this context, task allocation strategy on the basis of results generated in first two steps must provide high fidelity real time simulation and minimization of hardware requirements, since for some kind of simulator system such computer configuration must be duplicated in high number of copies. As already pointed out, in this system all program segments are known in advance as its time required for their execution and therefore static allocation concept will be appropriate one. If the execution times for each module and communication times between processors are known, then the problem of task allocation is to determine such distribution of modules on parallel processors which support real time simulation with minimal hardware requirements. The number of software modules which form one task which will be assigned to one microprocessor depends on arithmetic complexity of problem modules and real time requirements. Possible assignment can be several modules to one microprocessor, or several microprocessors to one module. This relationship depends predominately on the processing power of each processing element, complexity of discretized modules and their dynamics, intensity of interprocessor communication and so on. At the beginning of the procedure we start with allocation of task to first microprocessor on the basis of block level critical path method [28]. For some common frame time we add modules to first microprocessor until the sum of periods of implementation of these modules is less than or equal to this common frame time T. After that we are going on, with assignment of a tasks to the following microprocessors. So that for each microprocessor must be valid

$$\frac{\sum_{i=1}^{n} T_{imp}^{i}}{T_{disc}} \leq 1 \quad (19)$$

where T_{imp}^{i} denotes the period of implementation module i normalized on common frame time **T** and **n** denotes the number of modules assigned to each microprocessor. In this way it is necessary to continue with the process of task allocation for each succeeding microprocessor, until all modules have been allocated. If equation (19) is valid for all modules, single processor implementation of related problems is possible. With such an approach it is possible to reduce idle time of each microprocessor in multiprocessor configuration. But on the other side this approach increases time delay due to finite computation time. This delay may cause the effects of instabilities in closed loop simulation with included external hardware or in the operator training applications.

In the case of fast and complex modules it is very often necessary to assign several microprocessors to one module. Using the equation segmentation methods or some form of parallel predictor corrector algorithms or block implicit methods it is possible to achieve the speed of real time processing.

The level of granularity in the process of task allocation depends predominantly on the architecture of multiprocessor system, decomposition techniques, complexity of modules, their dynamic real time constraints, numerical methods used for discretization and so on. On one side very fast modules lead to the fine level of granularity, for example statement level or instruction level. Massively parallel processor in order to achieve the high efficient utilization of available processing power needs fine level of granularity. On the other side relatively slow modules lead to the high level of granularity preferred in single bus multiprocessor configuration. Different level of granularity leads to the different level of intensity in interprocessor communication. Communication time is not negligible specially in fine level of granulations since communication time is comparable with computation time. In such circumstances, task allocation optimized only on maximization of parallel operation would not be at all optimal one, when communication times are taken into account, especially if a large amount of data is to be a shared. Therefore parallel multiprocessor simulation requires detail consideration and analysis in order to reduce i.e. to minimize interprocessor communication. Waiting for intermediate results and delays during communication time decreases throughput per processor as the number of processors grows.

With careful decomposition schemes which follow physical topology of problems and with choices of suitable methods for numerical discretization and task allocation strategy, hardware requirements for real time simulation can be significantly reduced. It is therefore desirable to perform extensive non real time simulation and analysis in order to determine the best manner of allocating program modules and to achieve the

most cost effective solution.

4.4. Simulator design for one spinning missile system

The above presented procedure can be illustrated by the following example which requires simulator design that must provide real time simulation of one spinning missile according to the desired level of accuracy.

After linearization of 6-DOF model about corresponding nominal trajectory and suitable physical partitioning, block diagram form of one spinning missile can be shown by Figure 6.

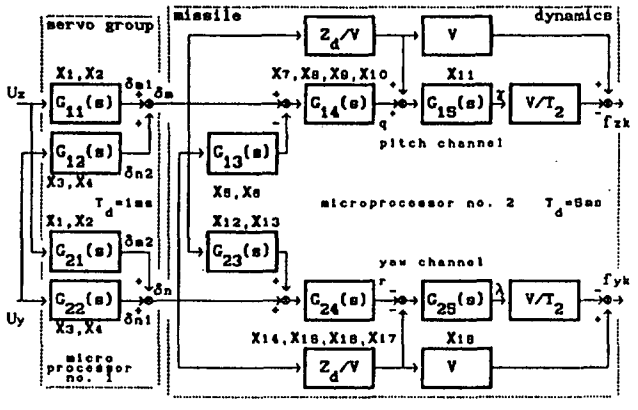


Figure 6. Block diagram of spinning missile in nonrotating coordinate system

Servo group mathematical model can be defined by the following system of ordinary differential and algebraic equations [29],

$$\begin{aligned}
\dot{x}_1 &= x_2 \\
\dot{x}_2 &= -1/T_p^2 \, x_1 - 2\xi/T_p \, x_2 + 1/T_p^2 \, U_z \\
\delta_{m1} &= K_p C_2 \, x_1 + K_p C_1 \, x_2 \\
\delta_{m2} &= K_p D_2 \, x_3 + K_p D_1 \, x_4 \\
\dot{x}_3 &= x_4 \\
\dot{x}_4 &= -1/T_p^2 \, x_3 - 2\xi/T_p \, x_4 + 1/T_p^2 \, U_y \\
\delta_{n1} &= K_p C_1 \, x_3 + K_p C_2 \, x_4 \\
\delta_{n2} &= K_p D_1 \, x_3 + K_p D_2 \, x_4
\end{aligned} \tag{20}$$

or in state space form with,

$$\begin{aligned}
\dot{x}_{pk} &= A_{pk} \, x_{pk} + B_{pk} \, U_{pk} \\
y_{pk} &= C_{pk} \, x_{pk}
\end{aligned} \tag{21}$$

In this case state, input and output vectors correspond to

$$x_{pk} = [x_1 \; x_2 \; x_3 \; x_4]^T, \quad U_{pk} = [U_z \; U_y]^T \tag{22}$$
$$y_{pk} = [\delta_m \; \delta_n]^T ,$$

while system matrices A_{pk}, B_{pk} and C_{pk} follow directly from (20).

Dynamics equations for pitch channel are given by

$$\begin{aligned}
\dot{x}_5 &= x_6 \\
\dot{x}_6 &= -1/T_0^2 \, x_5 - 2\xi/T_0 \, x_6 + 1/T_0^2 \, \delta_z \\
\delta_{mn} &= K_\sigma \, x_5 + K_\delta T_2 \, x_6 \\
\dot{x}_7 &= x_8 \\
\dot{x}_8 &= x_9 \\
\dot{x}_9 &= x_{10} \\
\dot{x}_{10} &= -a_1 \, x_7 - a_2 \, x_8 - a_3 \, x_9 - a_4 \, x_{10} + a_5(\delta_m - \delta_{mn}) \\
q &= K_q \, x_7 + K_q p_1 \, x_8 + K_q p_2 \, x_9 + K_q p_3 \, x_{10} \\
\dot{x}_{11} &= -1/T_2 \, x_{11} + (q + Z_\delta/V \, \delta_m) \\
\gamma &= x_{11} \\
f_{zk} &= V/T_2 \, \gamma
\end{aligned}$$

State space form of equation (23) is determined by

$$\begin{aligned}
x_{dz} &= [x_5 \; x_6 \; x_7 \; \cdots \; x_{11}]^T \\
u_{dz} &= [\delta_m \; \delta_n]^T \\
\dot{x}_{dz} &= A_{dz} \, x_{dz} + B_{dz} \, U_{dz} \\
f_{zk} &= C_{dz} \, x_{dz} + D_{dz} \, U_{dz}
\end{aligned} \tag{24}$$

Dynamic of yaw channel is identical and is given by,

$$\begin{aligned}
x_{dy} &= [x_{12} \; x_{13} \; x_{14} \; \cdots \; x_{18}]^T \\
u_{dy} &= [\delta_m \; \delta_n]^T \\
\dot{x}_{dy} &= A_{dy} \, x_{dy} + B_{dy} \, U_{dy} \\
f_{yk} &= C_{dy} \, x_{dy} + D_{dy} \, U_{dy}
\end{aligned} \tag{25}$$

Non real time simulation of the above equations has been performed on Cyber 170/850 using standard routine for numerical integration of ODEs from IMSL library (DVERK) in order to generate reference solution. With common period of integration of 1ms, and with 6-order Runge-Kutta methods reference solutions have been obtained and are shown in Figure 7.

Derivation of discrete time model for real time simulation requires careful choice of a method for numerical integration and period of discretization. For this linearized problem one of suitable numerical methods for discretization is the approach given by equation (4) and (5). After detailed dynamic analysis of original continuous system based on eigenvalues inspection and their distribution, the concept of multi rate processing has been accepted. Real time processing in servo group has been performed with 1ms period. In simulation of rigid body dynamic equations the period of discretization is 5ms. The complete discrete time model has been tested through extensive non real time simulation which are shown in Figure 7.

Program for testing discrete time model is based on equations (21),(24),(25) and (4) is given by

```
for i=1 to ....
*servo block
 for j=1 to 5
 input u_pk,k
 x_pk,k = A*_pk x_pk,k-1 + B°_pk (u_pk,k + u'_pk,k-1)
 y_pk,k = C*_pk x_pk,k
 output  y_pk,k(δ_m,k, δ_n,k)
 x_pk,k-1 = x_pk,k
 u_pk,k-1 = u_pk,k
 next j
*dynamic of pitch and yaw channel
 x_dz,k = A*_dz x_dz,k-1 + B°_dz (y_pk,k + y_pk,k-1)
 x_dy,k = A°_dy x_dy,k-1 + B°_dy (y_pk,k + y_pk,k-1)
```

$f_{zk,k} = C^{*}_{dz}\, x_{dz,k} + D^{*}_{dz}\, y_{pk,k}$

$f_{yk,k} = C^{*}_{dy}\, x_{dy,k} + D^{*}_{dy}\, y_{pk,k}$

output f_{zk}, f_{yk}

$x_{dz,k-1} = x_{dz,k}$

$x_{dy,k-1} = x_{dy,k}$

$y_{pk,k-1} = y_{pk,k}$

next i

Comparative analysis of results presented in Figure 7, shows a good tuning of discrete time model that has been provided with compensation matrices L=I and Γ=1/2I (bilinear transformation) and corresponding periods of discretization.

Strategy of static task allocation which follows physical decomposition of problem and choices of the numerical methods for discretization must meet requirements for real time simulation with minimal hardware requirements. Computational load i.e. speed up factor of first module-servo group is determined by period of discretization (1ms) divided by execution time of arithmetic operations needed for implementation of this discrete time model. The speed up factor computed for this module gives K_{servo}=1.0013. This means that real time simulation of this module is possible on only one microprocessor board and that the load balancing for this microprocessor is near ideal. The speed up factor for the following group which presents dynamics of pitch and yaw channel is $K_{dinamic}$=1.00124. This means that real time simulation

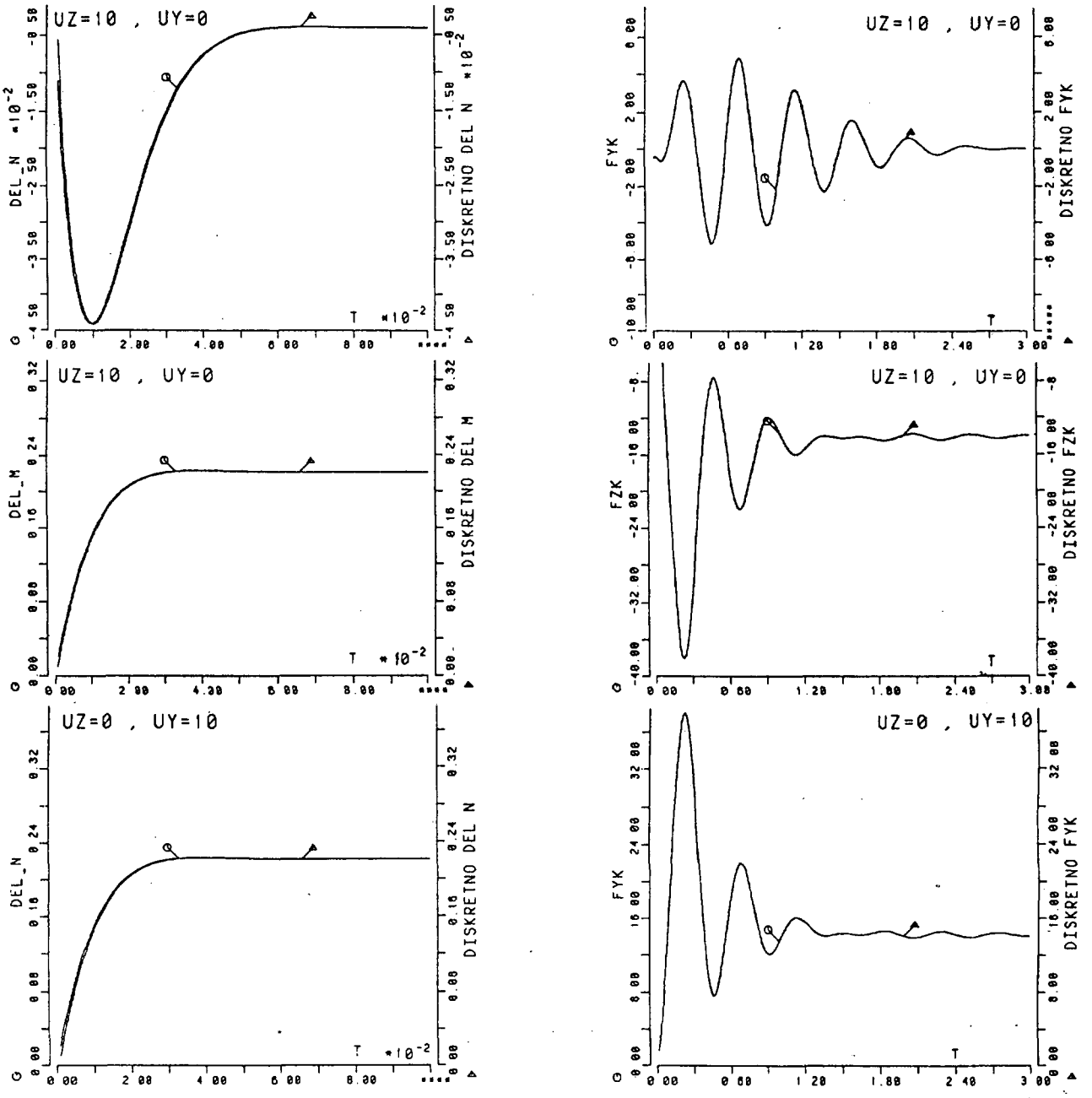


Figure 7. Time responses of continuous and equivalent discrete time models
(T_{servo}= 1ms, T_{dynam}= 5ms, L = I, Γ = 1/2 I)

of this module is also, possible and that it also needs only one microprocessor with near ideal load balancing. From the obtained results follows that the real time simulation of the model given in Figure 6, is possible with two near ideal equally load balance microprocessors. Assignment of tasks is determined by allocation of the discrete time model of servo group to microprocessor no.1, while pitch and yaw dynamic equation to microprocessor no.2. The intermediate results computed by first microprocessor must be transferred to the second one. However second microprocessor must check that intermediate results or sequence of computation and require synchronization between microprocessor no. 1 and microprocessor no. 2. To insure correct synchronization and acceptance of correct results, source microprocessor also broadcasts a ready flag with its results. The destination microprocessor i.e. the second microprocessor waits for ready flag before using the result broadcasted by the first microprocessor. Than it resets the ready flag after it detects that ready flag is set. Procedure executed by a source and destination microprocessor in transferring and waiting for exchange data and variables are standard in such cases.

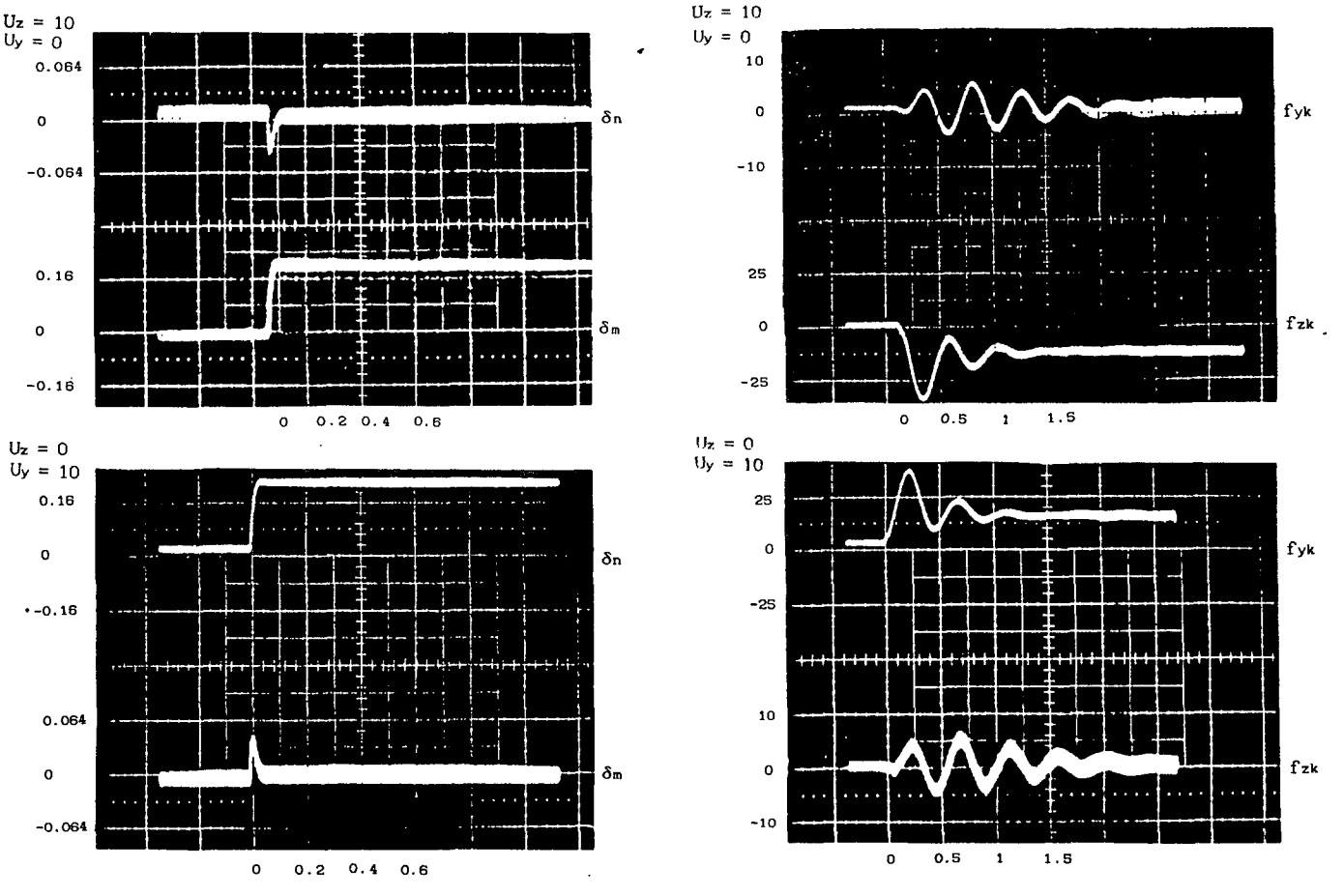


Figure 8. Real time simulation of one spinning missile system

data have been received before using them. In this example we have used for integration implicit method from accuracy and stability reason. But implicit integration methods are not suitable for parallel simulation, since they require careful consideration to insure correct synchronization between microprocessors. In execution of iteration k, first microprocessor must compute the output of servo block $y_{pk,k}$ and broadcast the results to second microprocessor as soon as possible. The second microprocessor in corresponding iteration for computation of its state vector and output equation must wait for values $y_{pk,k}$ from first microprocessor. These data dependencies that exist among servo block and dynamic block determine the

Program for parallel real time simulation for these two microprocessors is coded in assembly language according to the following relations.

```
*microprocessor no. 1 - servo block
*
*interrupt routine (real time clock - 8253 - 1ms)
*
 input u_pk,k          (from a/d converter over AMS-bus)
 x_pk,k = x^P_pk,k + B*_pk u_pk,k
 y_pk,k = C*_pk x_pk,k
 output  y_pk,k              (to microprocessor no.2
                              over dual port memory)
 x^P_pk,k = A*_pk x_pk,k + B*_pk u_pk,k
 idle                  (wait for next real time clock)
```

```
*
*microprocessor no.2 -dynamic of pitch and
*yaw channel
*
*interrupt routine (real time clock - 8253 - 5ms)
input y              (from microprocessor no.1 over
                     dual port memory - wait for
                     semaphor synchronization)
x_dz,k = x^P_dz,k + B*_dz y_pk,k
x_dy,k = x^P_dy,k + B*_dy y_pk,k
f_zk,k = C*_dz x_dz,k + D*_dz y_pk,k
f_yk,k = C*_dy x_dy,k + D*_dy y_pk,k
output f_zk,k, f_yk,k   (to d/a converters over
                         AMS - bus)
x^P_dz,k = A*_dz x_dz,k + B*_dz y_pk,k
x_dy,k = A*_dy x_dy,k + B*_dy y_pk,k
idle                 (wait for next real time clock)
```

The obtained real time solutions that satisfy the accepted objective function are shown in Figure 8 and can be considered as near optimal ones.

5. CONCLUSION

Presented methodology enables high efficient real time integration of complex dynamic system described by ordinary differential equations on multiprocessor system. It means that in the simulator design it is the most important to find the optimal combination of decomposition techniques, discretization algorithms and strategy of task allocation, that leads to the minimal number of microprocessors necessary for simulator realization in agreement with desired accuracy specifications. Through the attached peripheral homogenous single bus tightly coupled multiprocessor system, realized digital simulator provides a user with different experimental abilities in control system design, testing, modification and in the operator training.

6. REFERENCES

[1] W. H. Christal, D. G. Mackey: Guidance and Control Simulations for Laser Guided Weapons, Proceedings of the Conference on Aerospace Simulation, SCS, February 1984.

[2] K. Ćosić, S. Deskovski: Digitalna simulacija prostornog kretanja rotirajuće rakete u realnom vremenu, XXVIII konferencija ETAN-a, Split, 1984.

[3] K. L. Hall: A simulation system architecture for real time applications, The Proceedings of the Summer Computer Simulation Conference, 1986.

[4] K. Ćosić, S. Deskovski, I. Miler, M. Slamić : Digitalni simulator za razvoj i testiranje PO sistema vodenja, XXXI konferencija ETAN-a, Bled, 1987.

[5] J. W. Karplus: Computer hardware in the simulation, Lecture notes, ETH Zürich 1986.

[6] R. J. Hickman: Hardware-in-the-loop simulation of dynamics system with high performance multi -purpose digital computers, The Proceedings of the Summer Computer Simulation Conference, 1987.

[7] A. C. Watts: Aerospace system simulation at Sandia National Laboratories, The Proceedings of the Summer Computer Simulation Conference, 1987.

[8] R. A. Bleck, D. J. Arpasi: Hardware for a real-time multiprocessor simulator, The Proceedings of the Summer Computer Simulation Conference, 1985.

[9] R. Gluck: Hope for simulating flexible spacecraft, Aerospace America, November 1986.

[10] J. O. Hamblen: Parallel continuous system simulation using the Transputer, Simulation, December 1987.

[11] K. Owen: Rise of the supercomputer, Aerospace America, July 1988.

[12] D. Ghosal, L. M. Patniak: SHAMP: An Experimental Shared Memory Multiprocessor System for Performance Evaluation of Parallel Algorithms, Microprocessing and Microprogramming 19, 1987.

[13] E. Pearse O'Grady, C. H. Wang: Multibus-based parallel processor for simulation, Proceedings of the Conference on Aerospace Simulation, SCS, February 1983.

[14] K. Ćosić, I. Miler, D. Hadžiomerović: Koncept multiprocesorskog simulatora za testiranje sistema vodenja u realnom vremenu, Zbornik MIPRO 88, 1988.

[15] K. Ćosić, S. Deskovski, I. Miler, M. Slamić, I. Kopriva: Razvoj multiprocesorskog ekspertnog sistema za projektiranje sistema vodenja i upravljanja, XXXII konferencija ETAN-a, 1988.

[16] M. C. Gilliland, B. J. Smith, W. Calvert: HEP: A semaphore - synchronized multiprocessor with central control, Proceedings of the Conference on Aerospace Simulation, SCS, February 1976.

[17] H. Kirrmann: Events and Interrupts in Tightly Coupled Multiprocessors, IEEE Micro, February 1985.

[18] K. Ćosić, S. Deskovski: PC-based development system for simulator design, 3rd European Simulation Congress, Edinburgh 1989 (to be published).

[19] J. R. Pimentel; Real time simulation using multiple microcomputers, Simulation, March 1983.

[20] K. Ćosić, I. Kopriva: Sinteza diskretnih modela za digitalne simulacije u realnom vremenu, Automatika, br.5-6, 1987.

[21] K. Ćosić, I. Kopriva: Design of the optimal discrete time model for digital real time simulation, European Simulation Multiconference, Roma 1989.

[22] K. Ćosić: Design and implementation of discrete time model for real time digital simulation, All About Simulators, Simulators series, vol. 14, No.1, 1984.

[23] R. M. Howe: Special considerations in real time digital simulation, Summer Computer Simulation Conference, 1983.

[24] M. Franklin: Parallel solution of ordinary differential equations, IEEE Transactions on Computers, vol.c-27, No.5, May 1978.

[25] O. A. Palusinski: Simulation of dynamic systems using partitioning and multirate integration techniques, Summer Computer Simulation Conference, 1983.

[26] O. A. Palusinski: Simulation methods for combined linear and nonlinear systems, Simulation, March 1978.

[27] E. Pearse O'Grady, Chang-Hsein Wang; Parallel processor performance in a jet engine simulation, Summer Computer Simulation Conference, 1984.

[28] A. Makol, W. J. Karplus: ALI: A CSSL/multiprocessor software interface, Simulation, August 1987.

[29] S. Deskovski, M. Slamić: Matematički modeli i simulacije sistema upravljanja rotirajućih raketa, Naučno tehnički pregled, vol.XXXIV, br.1., 1984.

ISDN USER-NETWORK INTERFACE LAYER 3

INFORMATICA 3/89

Keywords: ISDN network, ISDN user-network inferface, ISDN user-network interface layer 3 protocol.

Gabi Dreo
Bogomir Horvat
Rado Slatinek
Tehniška fakulteta Maribor

Abstract: The Integrated Services Digital Network (ISDN) offers access to a wide range of services over a single subscriber line. An important part of the ISDN is the ISDN user-network interface. The main feature of the ISDN user-network interface is to support the various service capabilities, including voice and non-voice applications in the same network. Comprehensive interface standards are therefore very important for achieving universal deployment and worldwide compatibility. This paper presents the signalling procedures, according to Recommendation I.451/Q.931, for circuit-switched connections using the B-channel at the ISDN user-network interface layer 3, also mentioning the new features developed in these Recommendations, according to the CCITT Blue book. In the last section of the article a possible conceptual software structure of the lowest three ISDN layers is presented.

Povzetek: Mreža ISDN ("Integrated Services Digital Network") omogoča uporabniku dostop do različnih storitev (prenos govorne, podatkovne, tekstovne in slikovne informacije) preko ene naročniške linije. Pomemben del mreže ISDN je ISDN vmesnik uporabnik-mreža, ki omogoča dostop do njenih storitev. Enotni standardi vmesnikov so zato eden od pomembnih dejavnikov za dosego svetovno enotnega razvoja mreže ISDN. V članku je predstavljen protokol 3. nivoja v ISDN vmesniku uporabnik-mreža za tokokrogovno komutacijo in dvotočkovno povezavo po standardu I.451/Q.931 (CCITT Modra knjiga). Zadnji del članka je namenjen predstavitvi možne programske strukture prvih treh nivojev ISDN.

1 Introduction

The Integrated Services Digital Network (ISDN) is a telecommunication network composed of a wide range of network capabilities which are described by standardized protocols and functions /7/. The key to handle such an all-embracing coverage of services in a single network is service integration. This approach permits the ISDN to be viewed as a whole, regardless of whether only the part of the functions supported by the ISDN are needed for a certain service. So, the main progress, from the economical and technical point of view, is a single network which forms the basis for supporting various voice and non-voice applications, instead of having a separate network for each service.

Comprehensive interface standards are one of the key elements in achieving the universal deployment and worldwide compatibility. The ISDN will also be recognized by the characteristics observed at the user-network interface rather than by its internal arhitecture, configuration or technology. Therefore, interfaces are central to ISDN and determine most of ISDN's principal features.

To achieve the idea of integrated use of speech, data and image requires complete end-to-end digital transport, although the process of change is considered as an ongoing evolutionary process; or vice versa, digital information standardization leads to integration of the transmission, switching, and control functions-hence the concept of ISDN /2/.

We already mentioned two fundamental elements for the ISDN, digital connectivity for information transfer and the multipurpose capability of user-network interfaces. The third element is common channel signalling in which signalling information relating to a multiplicity of circuits or functions or for network managment is conveyed over a single channel by addressed messages /7/.

In our discussion we will first briefly describe the ISDN user-network interface. Our attention will be directed to the layer 3 protocol according to Recommendation I.451/Q.931 which contain the signalling procedures for establishing, modifying and terminating network connections /4,6/ across an ISDN between communicating application

entities.

Call control procedures may be divided into functional and stimulus procedures. Functional operation is more convenient for autonomous user equipment meanwhile stimulus operation is more appropriate for manually operated user equipment. Therefore our discussion will be directed to the functional call control procedures for the establishing and clearing of circuit-switched connections using the B-channel /4,8,9/.

2 General

2.1 ISDN user-network interface

An objective of ISDN is that a small set of compatible user-network interfaces can economically support a wide range of user applications, equipment and configurations (Fig.1.)/7/.

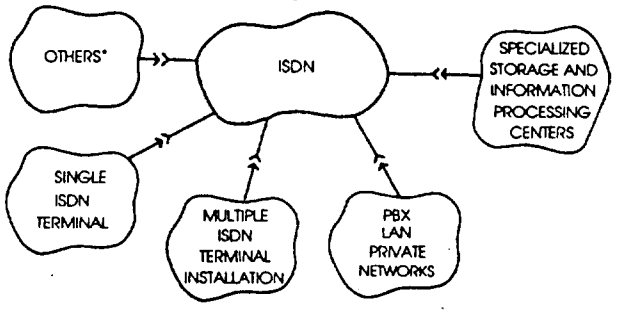
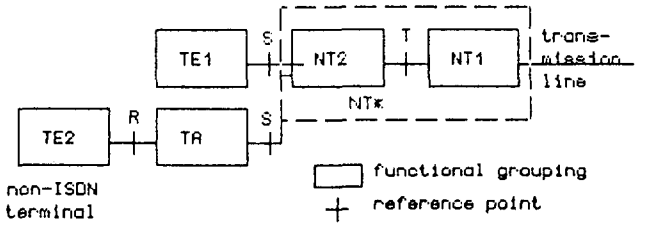


Fig.1. ISDN user-network interface examples

Fig.2. illustrates a more datailed view of the reference configurations for ISDN user-network interfaces /9/.



TE1-terminal equipment type 1 (digital telephones, data terminal equipment and integrated work stations)
NT2-network termination 2 (PABX, local area networks and terminal controllers)
TA-terminal adapter
NT*-NT2 and NT1 are combined as NT for a single interface

Fig.2. Reference configurations for the ISDN user-network interface

Each square represents a functional grouping which is a set of functions, needed in ISDN user access arrangements. The crosses represent the reference points which are dividing functional groupings. The reference points S and T are specially important because the ISDN user-network interface Recommendations in the I-series apply to physical interfaces at these reference points. Now we can also explain the term user-network interface as the interface between the TE and the NT. The TE is the equipment that provides the functions necessary for the operation of the access protocols by the user meanwhile the NT provides the functions necessary for the network access.

2.2 Communication contexts

Fig.3. will help us to understand the communication contexts for circuit-switched connections /5/. As we see, four communication contexts are shown:

- Context A is the realm of user-user information transfer over the circuit-switched connection.
- Context B is the realm of user-network signalling between the left TE (terminal equipment) and the ISDN.
- Context C is the realm of inter-network signalling.
- Context D is the realm of user-network signalling between the right TE and the ISDN.

There are three phases for establishing a call from a calling user to a called user:

a) the user-network phase (context B - signallization over the D-channel)
b) the internetwork phase (context C - the Signalling System No.7), and
c) the network-user phase (context D - signallization over the D-channel).

We will describe the establishing and clearing of a call in the user-network phase and touch the internetwork phase. The network-user phase is almost the same as the user-network phase, only with few modifications. The modifications depend on whether multipoint or point-to-point terminal configuration exist. If we carefully analyze the protocol blocks which represent the right and the left TE, we see that the user plane (U-plane) is a 7-layer hierarchical structure. The control plane (C-plane) is represented as a 3-layer structure because the higher four layers are used for user-user signallization, represented as context A. Context B and context D represent the signallization over the D-channel - the theme of our discussion. The internetworking phase (context C) is under an another signallization - the Signalling System No.7 meanwhile the user-user transfer (conext A) is a combination of both signallizations mentioned above - the signallization over the D-channel and the SS (Signalling System) No.7 /2,8,9/.

2.3 Concepts of layering

According to the layering technique (OSI), communication among application processes is viewed as being logically partitioned into an ordered set of layers (Fig.3.). The vertical communication between adjacent layers takes place with primitives, which represent, in an abstract way, the logical exchange of information and control /6/.

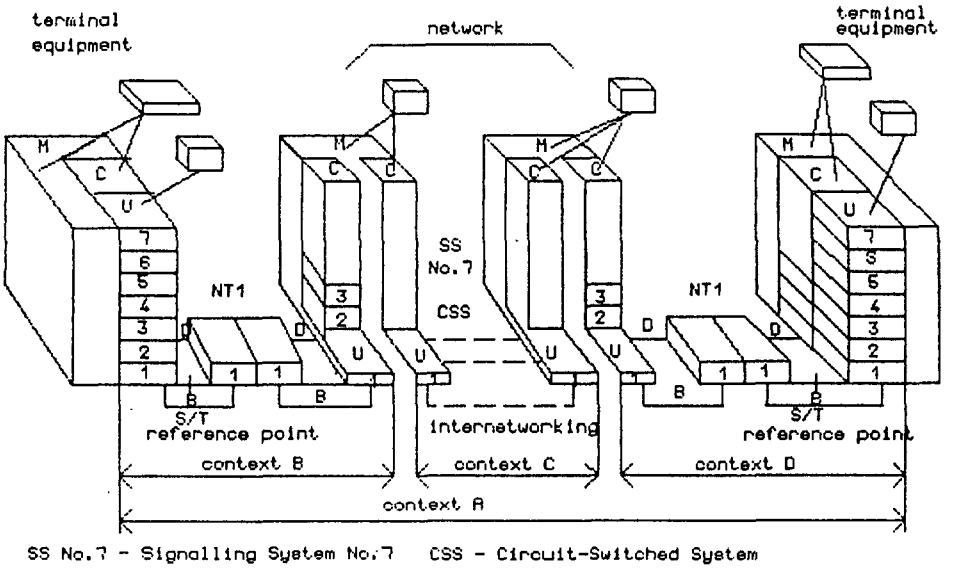


Fig.3. Communication contexts for circuit-switched connections via B-channel

Layer 3 provides to the user the functions associated with the establishment and operation of network connections. It also makes invisible how it utilizes underlying resources such as data link connections to provide a network connection. The main features of the layer 3 protocol can be summarized as follows /2,3/:

a) handling of a wide variety of connections through the same user-network interface which includes establishing and clearing of data link connections;
b) message structure consisting of elements common to all the message types;
c) symmetrical protocol for outgoing and incoming calls to enable direct user-to-user connection etc.

3 Signalling procedures

3.1 Call states

A call is, during the establishing, maintaining or clearing phase, in a specific state. The basic call control states are divided into call states at the user or the network side of the interface (Fig.4.)/8,9/.

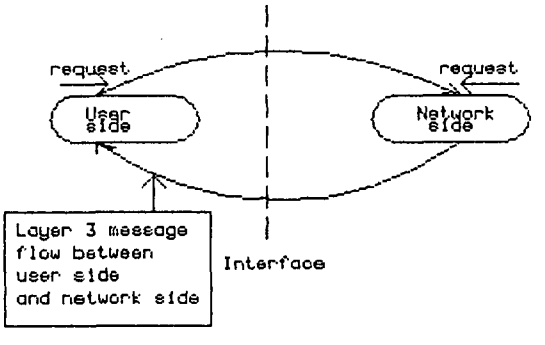


Fig.4. Convention for message transmission.

Because several calls may exist simultaneously at a user-network side it is obvious that the state of the interface itself cannot be unambiguously defined. Table 1 illustrates the user states (U) and the network states (N) /9/.

TABLE 1
User call states and network call states

User states	Network states
U0-null	N0-null
U1-call initiated	N1-call initiated
U2-overlap sending	N2-overlap sending
U3-outgoing call proceeding	N3-outgoing call proceeding
U4-call delivered	N4-call delivered
U6-call present	N6-call present
U7-call received	N7-call received
U8-connect request	N8-connect request
U9-incoming call proceeding	N9-incoming call proceeding
U10-active	N10-active
U11-disconnect request	N11-disconnect request
U12-disconnect indication	N12-disconnect indication
U15-suspend request	N15-suspend request
U17-resume request	N17-resume request
U19-release request	N19-release request
	N22-call abort
U25-overlap receiving	N25-overlap receiving

3.2 Message structure

The application entities communicate with the exchange of messages. Each message, according to the Recommendation I.451 /9/, includes:

a) a brief description of the message direction and use,
b) a table listing the information elements (mandatory or optional) contained in the message.

Table 2 lists the messages for circuit-mode connection control.

TABLE 2
Messages for circuit mode connection control

Call establisment messages	Call info. phase messages
ALERTing	MODIFY
CALL PROCeeding	MODIFY ACKnowledge
CONNect	MODIFY REJect
CONNect ACKnowledge	RESUME
PROGress	RESUME ACKnowledge
SETUP	RESUME REJect
SETUP ACKnowledge	SUSPEND
	SUSPEND ACK
	SUSPEND REJect
	USER INFOrmation
Call clearing messages	**Miscellaneous messages**
DISConnect	CONgestion CONtrol
RELease	NOTIFY
RELease COMPlete	INFOrmation
RESTart	STATUS
RESTart ACKnowledge	STATUS ENquiry

Within the Recommendation I.451, every message shall consist of the following parts (Fig.5.):
a) Protocol discriminator distinguishes I.451 call control nad maintaince messages from all other OSI network layer protocol (i.e. X.25 Packet Layer Protocol, etc.) on the D-channel.
b) Call reference means for tracing active calls and associated resources which are being utilized. The call reference value is assigned by the call originator ath the begining of the call and remains fixed for the life of the call.
c) Message type identifies the function of the message.
d) Mandatory and optional information elements which contain the required information.

The coding of information elements is formulated to allow each equipment to process the information elements important to it, and yet remain ignorant of information non-important to it.

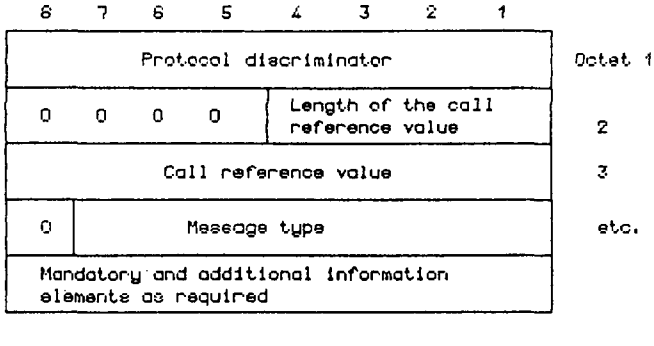


Fig.5. General message structure.

The information elements are the most important part of the message because they carry the user information meanwhile the protocol discriminator, call reference and the message type are used for control functions. There are two categories of information elements:
a) single octet information elements which can be everywhere in the message; and
b) variable length information elements with certain extension mechanisms.

For example, the bearer capability information element carries the information about the transfer mode (circuit, packet), information transfer rate, configuration (point-to-point, multipoint), information transfer capability etc. We have described only a part of the bearer capability information element to indicate the complexity and the numerous information carried in an information element.

3.3 Call establishment

After the user has initiated call establishment (similar as pick up the receiver), he enters data via a keyboard (the called party address). Layer 3 (of the terminal) sends now a SETUP message to the network (context B in Fig.3.) /1,4,8,9/. The SETUP message may contain all or part of the called party address, depending on whether en-bloc or overlap sending is being used respectively. If en-bloc sending is used, the SETUP message will contain all the information (complete called party address) required by the network to process the call. In case of overlap sending only a part of the information will be sent.

If the network determines, after receiving the SETUP message or during overlap sending, that the call information received from the user is invalid, it will initiate call clearing.

If the user terminal can monitor the status of channels in use, he shall not transfer a SETUP message across the user-network interface in case all appropriate channels controlled by the D-channel are in use. In the opposite case, when the user terminal does not monitor the status of channels in use, he may send a SETUP message during an all channels busy condition. The network responds with a RELease COMPlete message.

In the SETUP message the user terminal can indicate one of the following:
a) channel is indicated, no acceptable alternative; or
b) channel is indicated, any alternative is acceptable; or
c) any channel is acceptable.

If no indication is included, alternative (c) is assumed. The selected B-channel is indicated in the first message returned by the network. If the specified channel in case (a) is not available, or no channel in cases (a) and (b), a RELease COMPlete message with an appropriate cause is sent by the network.

The terminal selects a call reference for the call which is unique within the scope of a call. In case of en-bloc sending the network will send a CALL PROCeeding message to the user terminal to indicate that the call is being processed.

If overlap sending is used, the SETUP message contains either:

a) no called number information; or
b) incomplete called number information; or
c) called number information which the network cannot determine to be complete.

On receipt of such a SETUP message, the network sends a SETUP Acknowledge message to the user. After receiving the SETUP Acknowledge message, the user terminal sends the remainder of the information (if any) in one or more INFOrmation messages. An INFOrmation message may contain, besides the possible called party number, also aditional call information (i.e. for supplementary services). After the network has received a sending complete indication or has analyzed that all necessary call information has been received, it shall send a CALL PROCeeding message. The CALL PROCeeding message indicates that access to the requested service and supplementary service is authorized and available.

The originating exchange will transmit the call through the network to the destination exchange and then to the called terminal. During call establishment, the call may leave an ISDN environment because of interworking with a non-ISDN network (context C in Fig.3.). The calling user terminal shall be informed with a SETUP ACKnowledge/CALL PROCeeding/ALERTing/CONNect message and change the state of a call or with a PROGress message when no state change is appropriate. If the interface at which the message originates is the point at which a call enters the ISDN environment from a non-ISDN environment, appropriate progress indicator information elements shall be included in the SETUP message sent to the network.

After receiving the SETUP message, the called terminal will check the compatibility and respond with an ALERTing message. The called user will be informed with ringing (similar as the telephone ringing), and a ring back indication will be sent back to the calling user (analogy with the telephone - the calling user hears the ringing at the called party). If the called user answers (e.g.,picks up the telephone), a CONNect message will be sent to the destination exchange and then through the network to the originating exchange and the calling user. Now the connection is established and the call at both sides of the interface enters the Active state or the maintaining phase of the call. To confirm that the call is in the maintaining phase, a CONNect ACKnowledge message is sent from the originating to the destination exchange.

During the maintaining phase, the user notification procedure allows the network or the user to notify, by sending a NOTIFY message, the remote user of any call-related event. Another possibility, during the maintaining phase, is also to change the bearer capability for a call with a MODIFY message.

Upon receiving an indication that the network or the called user is unable to accept the call, the network shall initiate call clearing.

3.4 Call clearing

Under normal conditions, call clearing is usually initiated by the user or the network. We will only explain the call clearing initiated by the user (calling or called user hangs up). The signalling procedures for call clearing initiates by the network are almost the same.

The terminal forms and transfers a DISConnect message across the user-network interface, starting a timer, disconnecting the B-channel and entering the Disconnect Request state. After the network has received the DISConnect message, the B-channel used in the call is disconnected and a RELease message is sent back to the user and a timer is started. On receipt of the RELease message, the user shall cancel the started timer, release the B-channel and the call reference, and send in response a RELease COMPlete message. After the network has received the RELease COMPlete message from the user, it shall stop the started timer and release both the B-channel and the call reference. If the timer, started by the user, expires, the user shall again send to the network a RELease message with the cause number originally contained in the DISConnect message; start again the same timer and enter the Release Request state. The network will, after the expiry of its timer, again send a RELease message and restart the timer. After the second expiry of the timer (e.g., no RELease COMPlete message was received), the network shall place the B-channel in a maintaince condition, release the call reference and return to the Null state. The B-channel in the maintaince condition can be activated with the restart procedure.

5 A possible software implementation of the first three layers of the ISDN

Fig.7. illustrates a possible task based conceptual software solution of the ISDN. The principle is similiar to the concept of the MINIX operating system. The first layer is a compound of software drivers which have to manage the appropriate hardware unit. Therefore software drivers are strictly tied to the hardware structure. The adjacent higher layer consists of tasks. They do not have to know the physical structure of the hardware but only the logical functions of every hardware unit. Managers are the logical highest units which communicate with the hardware with the assistance of tasks and handlers. The hierarhy concept is similar to the TOP DOWN design. The basic program structure of all three types of units is the same: they operate in an endless loop. An example of the pseudo-cod is represented.

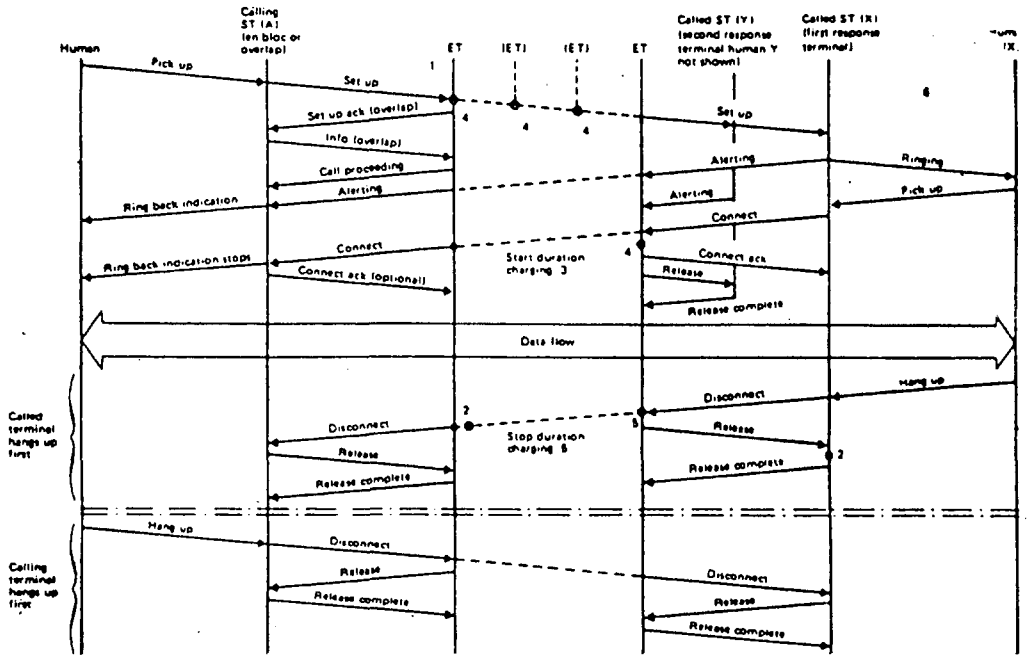


Fig.6. Call establishment and release

```
repeat
  wait_for_request;
  case request of
    req_1: actions_1;
    req_2: actions_2;
        .
        .
        .
    req_n: actions_n;
  end;
forever;
```

When a logical unit (handler, task, manager) executes the procedure wait_for_request and no request is present the process (executive unit) is suspended - it is in the WAIT state and so the process is not burdening the processor. The process can be reactiveted with a software (from a higher) or hardware interrupt (from a lower layer).

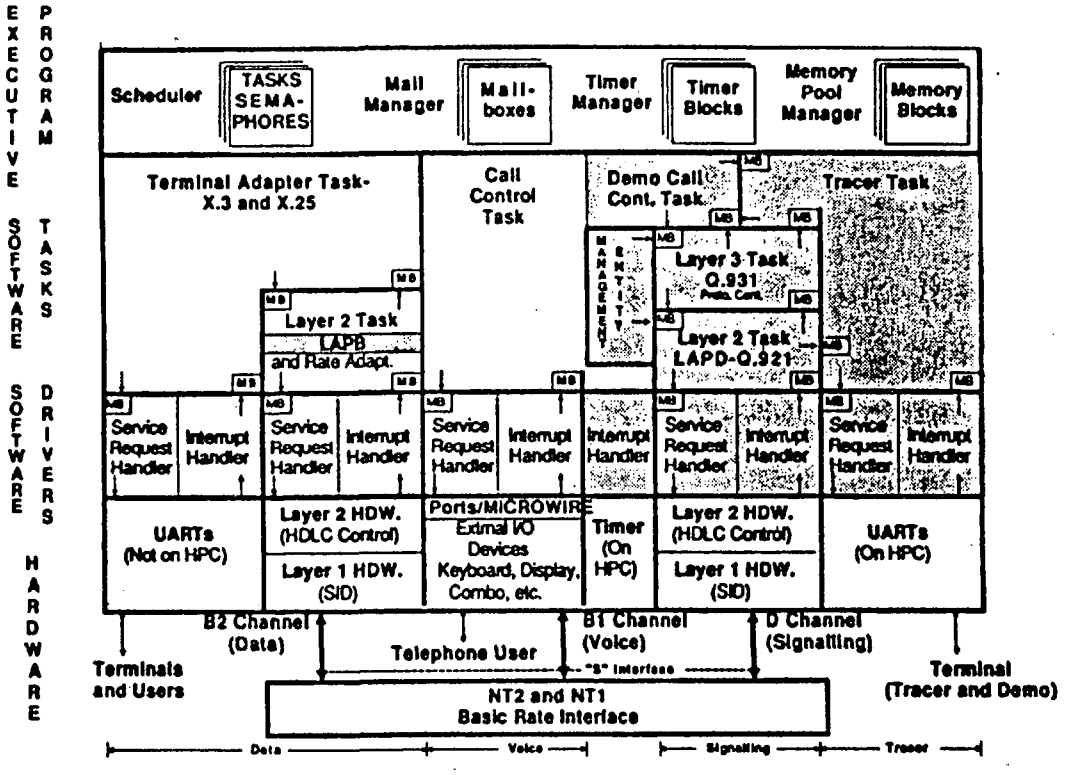


Fig.7. Conceptual software structure for the first three layers of the ISDN

6 Conclusion

Over the years, the concept of the ISDN has evolved from a vision to its technical realization. The current state of a part of the ISDN user-network interface layer 3 has been presented in this paper. We have concentrated on the signalling procedures for establishing, maintaining and clearing of circuit-switched connections using the B-channel, according to Recommendation I.451/Q.931, Blue book. In the last part of the article a conceptual program solution for the three lowest layers of the ISDN has been presented.

References

/1/ T.O'Toole, "Aufbau eines ISDN-Terminals", Elektronik, Vol.3, pp.129-134, Muenchen, March 1989.

/2/ IEEE Journal on Selected Areas in Communications, vol.sac-4, No.3, May 1989.

/3/ CCITT Recommendation I.450/Q.930, "ISDN user-network interface layer 3-General aspects," McGraw-Hill's Compilation of Data Communications standards, Vol.1.

/4/ CCITT Recommendation I.451/Q.931, "ISDN user-network interface layer 3-Specification", McGraw-Hill's Compilation of Data Communications standards, Vol.1.

/5/ CCITT Recommendation I.320, "ISDN protocol reference model", McGraw-Hill's Compilation of Data Communications standards, Vol.1.

/6/ CCITT IXth Plenary Assembly, "Report of study group XI to the CCITT Plenary Assembly, part III.14: Digital subscriber signalling No.1 (DSS1), Data Link Layer (R. Q.920 and Q.921)", Document AP IX-123, Dokument AP IX-124, Melbourn 1988.

/7/ A.M.Rutkowski, "Integrated Services Digital Networks", Artech House INC., 1985.

/8/ R.Slatinek, "D-channel layer 3 protocol on ISDN access (in Slovene)",Tehniška fakulteta, Maribor 1988.

/9/ R.S.Kole and C.Stacey, "Integrated Services Digital Networks, Protocol and Implementation", Frost and Sullivan - Seminar, London 1988.

/10/R.Slatinek, "Voice terminals concentrator and ISDN network", to be published.

This work was supported by Research Communities of Slovenia and ISDN Telematika.

INFORMACIJSKI PRINCIPI IN FORMALIZACIJA[1]

INFORMATICA 3/89

Keywords: informational principles, formalization of principles, informational formulas

Anton P. Železnikar[2]

Ta spis obravnava principe in iz njih izhajajoče posledice in primere t.i. informacijskih oblik in informacijskih procesov. Kot temelj razprave se konstruirajo informacijski principi, ki obsegajo informacijsko spontanost, pojem informacije in protiinformacije, za njimi pa še več principov informacijskega procesiranja, kot so informiranje, vmeščevanje, nastajanje in protiinformanje informacije. Opredeljujejo se t.i. cirkularne lastnosti informacije in sicer cirkularnost, rekurenca, vzporednost in zaporednost (posledičnost) informacije. Informacijska oblika in informacijski proces sta opisana z uporabo informacijskih principov in informacijskih posledic in kot posebna, informacijsko strukturirana in organizirana pojava (informacijsko postavje). Več informacijskih oblik in informacijskih procesov je prikazanih s primeri v obliki formul, ki kažejo na možnosti mehke, vendar formalno striktne formalizacije informacijskih konceptov. Raziskujejo se primeri inteligence kot informacije, informacijskega stroja, informacijskega programa in žive informacije, o katerih teče razprava v okviru že razvitih principov informacije. Na koncu je dodana kratka razprava o možnostih striktne matematične formalizacije (teorije) informacijskih principov. Za posamezne principe, posledice in primere, zapisane v verbalni obliki, so izpeljane tudi pripadajoče simbolne formule, ki kažejo na možnosti splošne formalizacije informacijskih principov.

INFORMATIONAL PRINCIPLES AND FORMALIZATION. This essay deals with principles, consequences, and examples of informational forms and informational processes. First, the principles of informational spontaneity, information, and counter-information are posted and then several principles of information processing are revealed: informing, embedding, arising, and counter-informing of information. The so-called circularity properties of information for these principles are investigated, which explain the circularity, recurrence, parallelness, and serialness (sequentialness) of information. Informational form and informational process are described on the basis of informational principles and informational consequences, and as particularly, informationally structured and organized phenomena (informational enframing). Several informational forms and informational formulas are presented as examples, explicating the possibilities of a soft informational formalization. Examples of intelligence as information, of an informational machine, an informational program, and living information are examined and discussed in this framework of developed principles of information. At the end, a short conclusion concerning strict formalization (theory) of informational principles is given. To given principles, consequences, and examples explicated in a verbal form, adequate symbolic formulas are derived, pointing out some possibilities of a general formalization of informational principles.

[1] Snov tega spisa je bila v svoji osnovni obliki (brez simbolično formalnega dela) objavljena pod naslovom "Principles of Information" v časopisu Informatica 11 (1987) 3, 9-17 [s popravkom v Informatica 11 (1987) 4, 26] in v časopisu Cybernetica (1988) 3, 99-122 v angleškem jeziku. Pričujoči spis dodaja formalni (ali formulski) del, v prevodu pa je prilagojen možnostim slovenskega jezika.

[2] Anton P. Železnikar, Iskra Delta, Stegne 15C, Ljubljana.

... Osnovni pogoj uporabe jezika je, da je poslušalec pripravljen vstopati v konverzacijo z govorcem. Pri tem ni pomembna količina logičnega ali lepega ali vznemirjajočega govorjenja, če to ni poslušano. Brezsmiselno je govorjenje, če sta ozadji govorca in poslušalca preveč različni za oblikovanje odprtega in resnega poslušanja; celo če se besede berejo, bo lahko njihova reinterpretacija preveč oddaljena.

Winograd [OUC] 257

UVOD

Eden od namenov tega spisa je odpiranje pogovora, s katerim bi bilo mogoče artikulirati spremenjene možnosti pojmovanja informacijske pojavnosti in s katerim naj bi se odkrivale posledice in znanstvene in tehnološke koristi predloženih principov. Ta konverzacija in njena moč naj bi bili utemeljeni z informacijsko fenomenologijo, utemeljeno z informacijskimi principi in posledicami teh principov. Informacijske principe in njihove posledice v tem spisu je mogoče razumeti tudi kot nastajajoči začetek, promocijo, hipotezo in temelj duhovno in arteficialno nastajajočega ali tudi kot perspektivno odprtost razvijajoče se filozofije informacije.

V splošnem velja, da so naravni principi (logično, ontološko) nekonsistentni. Ali velja to tudi za predložene informacijske principe? Na določen način vselej uporabljamo spretnost proste izbire, ko se igramo s formalnimi ali celo metafizičnimi pojmi in njihovimi simboli. Principi so le konstrukcije in realnost mora biti konstruirana, saj ne čaka, da bi jo odkrili [SCC].

Termin "informacija", njegov pojem, pomen in njegovo spoznavanje se navadno razumejo kot dani (pri-ročni) in zaradi tega ne vstopajo v razumevne (perceptivne in spoznavne) podrobnosti informacijske fenomenologije. Na določeni stopnji razvoja zavesti postane pojem informacije tudi procesiranje (informiranje), komuniciranje (interakcija) in informacijski sistem, še posebej v kontekstu nevralnih znanosti (nevrolingvistika, nevrofiziologija), socialna komunikacija (sociologija, lingvistika), računalniško možne interakcije človek-stroj (umetna inteligenca, nevralne mreže), informacijske uporabe (ekonomija) in raziskav v kontekstu kognitivnih znanosti (psihologija, psihiatrija).

Z vpraševanjem o principih informacije se lahko razvijajo nova vprašanja, ki povzročajo nove odgovore z zadostno splošno, naravno jezikovno in formalno strogostjo. V tem spisu ne bo pokazana epistemologija pojma informacije, ki je bila delno obravnavana v spisu [OWI], podrobneje pa še v spisu [IDI]. Informacija ima semantično cirkularen in dinamično spontan pomen in ta pomen je mogoče spremljati skozi zgodovino človeka (npr. od grškega "eidos", "morfe" in "poiesis" do latinskega "informatio" in do današnjega občega pomena informacije, informiranja, informativnega, informacijskega, procesno informacijskega itd.).

V pričujočem besedilu bo sistematično in jezikovno formalno razvitih nekaj osnovnih principov informacije. S tem razvojem bo omogočeno artikuliranje in izpostavitev informacijske pojavnosti kot dovolj splošnega izkustva zdravega razuma in dokaj regularnega principa (praktične oblike) sodobnega mišljenja. To mišljenje se utemeljuje in vmešča s spontanim in cirkularnim razumevanjem (rekurentnim opazovanjem, raziskovanjem in spoznavanjem). V prihajajoči informacijski dobi se bo razvilo informacijsko mišljenje in razumevanje skozi informacijo tudi s posebnim namenom strukturiranja in organizacije nove filozofije in uporabe novih (inteligentnih) tehnoloških možnosti. Na poti k informaciji se prav gotovo skrivajo novi pojavi, ki jih bo mogoče podrobneje raziskovati.

INFORMACIJA IN PROTIINFORMACIJA

... Vsaka zmagovita teorija preživi tri obdobja: najprej je odklonjena kot neresnična; potem je zavrnjena kot nasprotje religije; končno je sprejeta kot dogma in vsak znanstvenik poudarja, kako je že dolgo cenil resnico.

Gould [ESD], ko citira embriologa Von Beara

Informacijo izkušamo kot spontano razvijajoči miselni pojav našega uma. Ta izkušnja nam govori, da informacija ni samo razvijanje, spontano prihajanje v bivanje in nastajanje, temveč je tudi izginjanje, spreminjanje itd. V teh okoliščinah se informacija pojavlja kot skrajno svoboden in dinamičen proces našega uma. To raznovrstno informacijsko procesiranje je neomejeno cirkularno glede na celotno informacijo bitja. Vobče privzemamo, da informacija informira spontano in da se na svoji poti (razvoja) sooča s svojim lastnim opazovanjem, raziskovanjem in spoznavanjem, kar vse ima za posledico novo nastalo informacijo, ki jo imenujemo protiinformacija. Na poti k informaciji (lahko) postane protiinformacija regularna informacija zaradi vmeščanja protiinformacije v obstoječo informacijo; potem informacija kot informacija spet informira na opisani način.

Informacija kot informacija informira v različnih domenah. Bistvena domena informacije je očitno t.i. domena razumevanja, v območju katere informacija opazuje, raziskuje in razpoznava sebe in drugo informacijo. Skladno s to razumevajočo informacijsko komponento (domeno) lahko informacija rojeva nov, kontroverzen del same sebe in druge informacije, ko nasprotuje celotni obstoječi informaciji in povzroča pojavljanje protiinformacije. V tem pomenu ni nastajajoča protiinformacija ničesar drugega kot razumevajoča informacija informacije. Vobče je informacija informacijska in hkrati protiinformacijska. Ker je informacija samorazumevajoča, je tudi protiinformacijska. V svojem informacijskem obsegu je informacija tudi neinformacijska oziroma se nahaja izven obstoječe informacije, ko rojeva protiinformacijo.

Upoštevaje to okvirno informacijsko pojavnost je mogoče oblikovati troje začetnih principov informacije.

Princip informacijske spontanosti

Informacija je cirkularno spontani pojav informacije. Cirkularna informacijska spontanost je informacija tudi sama. □

S tem principom je mogoče informacijsko cirkularno spontanost razumevati kot osnovno informacijsko lastnost, ki ni še na noben način omejevana, vezana, pogojevana ali preprečevana informacija in je nepredvidljivo svobodna v svojem nastajanju, vmeščanju, protiinformiranju in cirkuliranju. Spontanost se tako nahaja izven in nad katerim koli avtopoetičnim (samoprodukcijskim) sistemom [AAC], ki je avtopoetično konsituiran. Navadno se spontanost razumeva tudi kot kaos ali priložnost, kot kaotično dinamičen, naključen ali neregularen pojav, kot turbulenten pretok, nepredvidljiva evolucija (bioloških vrst, ekonomskih cen), kot nereverzibilna evolucija [CCH] itd.

V spontanem se nahaja zgoščeni pomen prihajanja v bivanje, volativnega, svobodnega, nepričakovanega. Biti spontano je nemogoče brez prihajanja v bivanje nečesa, kar prihaja, kar se dogaja. Spontano ima pomen zaenkrat še neznanega, zakaj in kako bo spontano nastalo, celo kadar se ugotavlja, da je nekaj bilo spontano. "Je bilo" že ni več spontano, je svojo spontanost izgubilo takoj, ko je postalo ugotovitev. Spontano kaže v prihodnost, za katero ni vnaprej dano, kaj se bo pojavilo, zgodilo, nastalo, prišlo v bivanje, protiinformiralo itd. Spontano kaže v "bo", toda v sedaj neznano "bo". V tem pomenu se spontanost npr. ne sooča z znanostjo (ali celo z ideologijo), saj je znanost lahko le to, kar je znano. Na ta način znanost ne more (ne sme) biti spontana. V nasprotju z znanostjo pa je prvi princip informacije informacijska cirkularna spontanost, ki presega kateri koli princip resnice, krive vere, pravilnosti, identitete, objektivizma, tradicionalizma itd. Spontanost kaže v odpiranje, slutenje obzorja, v zavest, ki sega izza obzorja vedenja (znanega), v odpiranje k novim obzorjem in ostaja tako vselej odprta.

Informacijska spontanost je cirkularna. To kar spontano nastaja kot protiinformacija, se vmešča v obstoječo informacijo in iz te informacije spet napreduje (informira) kot spontano nastajanje informacije v obliki posledičnega procesa. Tako je delovanje spontanosti kot informacije cirkularno. Če informacijska spontanost označuje neko informacijsko lastnost (obliko) spontanosti, potem nastajanje označuje nek informacijski proces spontanosti, s katerim nastaja (prihaja v bivanje) informacija na spontan način.

Formule informacijske spontanosti

V okviru principa informacijske spontanosti se srečujemo s tremi osnovnimi pojmi, in sicer: informacija, spontanost in cirkularnost informacije. Označimo informacijo kot operand s simbolom α. Hkrati se dogovorimo, da bomo informacijske operande označevali z malimi grškimi črkami, tj. z α, β, γ, ... Spontanost in cirkularnost informacije α bosta implicitno zajeti v operatorju $\models$, ki bo splošen informacijski operator, imenovan tudi metaoperator in bo označeval proces informiranja oziroma informiranje. Operator $\models$ bo eksplicitni operator informiranja. Da bo splošnost formalne izražave še večja, bomo uvedli še simetrični operator informiranja $\dashv$, tako da bomo v primeru operatorjev $\models$ in $\dashv$ lahko rekli, da informira $\models$ na en, $\dashv$ pa na drug način. Princip informacijske spontanosti, se glasi: "Informacija je cirkularno spontani pojav informacije. Cirkularna informacijska spontanost je informacija tudi sama." Ta princip zapišemo formalno takole:

$$\alpha =_{Df} ((\alpha \models) \vee (\models \alpha)) \vee ((\dashv \alpha) \vee (\alpha \dashv))$$

Definicijska ekvivalenca $=_{Df}$ označuje definicijski konstrukt na desni strani znaka za njen levi označevalec. Ta definicija pravi, da informacija α informira tako ($\models$) ali drugače ($\dashv$) na spontan in cirkularen način. Spontanost je zajeta že v operatorjih $\models$ in $\dashv$, cirkularnost pa tudi v sami krožnosti definicije kakor tudi v paru $\alpha \models$ in $\models \alpha$ za en način v paru $\dashv \alpha$ in $\alpha \dashv$ za drug način informiranja. Odtod je namreč mogoče izpeljati t.i. aksiomatično krožnost, in sicer z implikacijama

$$((\alpha \models) \vee (\models \alpha)) \Rightarrow (\alpha \models \alpha) \quad \text{in}$$
$$((\dashv \alpha) \vee (\alpha \dashv)) \Rightarrow (\alpha \dashv \alpha)$$

Iz teh formul bi res sledila cirkularnost informacije α v obliki formule

$$\alpha =_{Df} ((\alpha \models \alpha) \vee (\alpha \dashv \alpha))$$

ki ni krožna le definicijsko temveč tudi rezultatsko v svojem desnem delu. Oklepaja '(' in ')' sta običajna separatorja oziroma mejnika, logična znaka (operatorja) $\vee$ in $\Rightarrow$ pa označujeta disjunkcijo in implikacijo. Na koncu verbalnih principov, posledic in primerov postavljamo znak '□', na koncu formalnih opisov pa znak '■'. ■

Princip informacije

Informacija informira tako, da cirkularno in spontano rojeva informacijo kot svojo protiinformacijo, da cirkularno in spontano vmešča novo informacijo v obstoječo informacijo, ali da informacija cirkularno in spontano nastaja, prihaja v bivanje, se spreminja, izginja, vmešča itd. s samo informacijo. Informacija ima lastnost obvladovanja (oblikovanja in procesiranja) same sebe. □

Na podlagi tega principa je mogoče razvijati filozofijo ali teorijo informacije v podrobnejši obliki. Iz tega principa izhaja ugotovitev, da predstavlja narava cirkularnega in spontanega oblikovanja in procesiranja informacije nek (matematično, algoritmično, formalno in tudi jezikovno) nenavaden (racionalno nelogičen) problem govorjenja in formalizacije, ki se nahaja (navidezno) izven t.i. racionalistične tradicije (npr. jeznega ali zadrtega zdravega razuma).

Formule informacije

Že v prejšnjem formalnem opisu spontanosti smo zapisali dve definicijski formuli informacije, in sicer

$$\alpha =_{Df} ((\alpha \models) \vee (\models \alpha)) \vee ((\dashv \alpha) \vee (\alpha \dashv)) \quad \text{in}$$
$$\alpha =_{Df} ((\alpha \models \alpha) \vee (\alpha \dashv \alpha))$$

S tema formulama smo potrdili načelo spontanosti in krožnosti. Pri tem velja poudariti, da sta tudi sami definicijski formuli informacijski, da ohranjata načelo spontanosti in krožnosti. Informirata pa tudi sami po sebi, torej sta informacija. Tudi vsi v teh formulah nastopajoči operatorji so informacijski (tipa $\models$ in $\models\!\!\dashv$), le da so partikulazirani na znane logične operatorje.

Skladno s prejšnjim principom informacije moramo k tema formulama dodati še načelo nastajanja (rojevanja, prihajanja v bivanje, spreminjanja, izginjanja) in vmeščanja nove informacije in protiinformacije. Označimo protiinformacijo s črko ω. Kadar protiinformacija ω nastaja iz informacije α, lahko uporabimo še oznaki ω_α ali $\omega(\alpha)$. Prvo oznako preberemo protiinformacija ω informacije α, drugo pa kot protiinformacija ω, ki je funkcija informacije α. Tako lahko zapišemo formalno

$$(\omega_\alpha \vee \omega(\alpha)) \equiv ((\alpha \models \omega) \vee (\omega \models\!\!\dashv \alpha))$$

To formulo preberemo takole: protiinformacija ω informacije α, ki jo označujemo kot ω_α ali $\omega(\alpha)$, je logično ekvivalentna (znak $\equiv$) nastajanju protiinformacije ω iz informacije α na en (operator $\models$) ali drug način (operator $\models\!\!\dashv$). Nastajanje informacije, ki je zajeto v operatorjih $\models$ in $\models\!\!\dashv$ (tudi v okviru samega pojma informacije α), ni le nastajanje v dobesednem pomenu, temveč je tudi rojevanje, prihajanje v bivanje, spreminjanje in izginjanje informacije. Tako nam preostane še pojasnilo o vmeščanju nove informacije in protiinformacije.

Vmeščanje $\mathfrak{E}$ druge informacije ali zunanje informacije β v informacijo α nastaja v okviru informiranja $\mathfrak{I}$ informacije α. Informiranje in vmeščanje informacije c sta lahko tako implicitni ($\mathfrak{I}_\alpha$ in $\mathfrak{E}_\alpha$) kot eksplicitni operaciji ($\models_\alpha$, $\models_{\mathfrak{I}}$, $\models_{\mathfrak{E}}$, $\models\!\!\dashv_\alpha$, $\models\!\!\dashv_{\mathfrak{I}}$, $\models\!\!\dashv_{\mathfrak{E}}$). Implicitne operacije ali funkcije v okviru informacijskih operandov bomo označevali z velikimi gotskimi črkami. Velika gotska črka bo tako vselej označevala neko obliko informiranja (informacijskega procesa nad informacijo kot operandom). Zunanja informacija β se vmešča v informacijo α že z njenim informiranjem, ko velja

$$((\beta \models \alpha) \vee (\alpha \models\!\!\dashv \beta)) \Rightarrow ((\beta \models_{\mathfrak{E}(\alpha)} \alpha) \vee (\alpha \models\!\!\dashv_{\mathfrak{E}(\alpha)} \beta))$$

To formulo preberemo takole: informacija β se lahko vmesti v informacijo α na en ali drug način le do stopnje, kot jo dopušča vmestitvena nastajalnost ($\models_{\mathfrak{E}(\alpha)}$ in $\models\!\!\dashv_{\mathfrak{E}(\alpha)}$) informacije α. V informiranju zunanje informacije β informacije α je treba upoštevati dvoje: prvič, zunanja informacija β informira tako, kot vobče informira vsaka informacija, torej $\beta \models$ ali $\models\!\!\dashv \beta$; drugič, informacija α je lahko informirana tako, kot je lahko informirana vsaka informacija, torej $\models \alpha$ oziroma $\alpha \models\!\!\dashv$. Dejansko imamo opraviti z operatorsko kompozicijo (znak $\circ$), tako da veljata v primeru, ko informacija β informira informacijo α, naslednji ekvivalenci:

$$(\beta \models \alpha) \equiv (\beta \models_\beta \circ \models_\alpha \alpha)$$
$$(\alpha \models\!\!\dashv \beta) \equiv (\alpha \models\!\!\dashv_\alpha \circ \models\!\!\dashv_\beta \beta)$$

Namesto teh ekvivalenc lahko za ta primer zapišemo še dve smiselni ekvivalenci, in sicer

$$(\beta \models \alpha) \equiv ((\beta \models_\beta (\models_\alpha \alpha)) \vee ((\beta \models_\beta) \models_\alpha \alpha))$$
$$(\alpha \models\!\!\dashv \beta) \equiv (((\alpha \models\!\!\dashv_\alpha) \models\!\!\dashv_\beta \beta) \vee (\alpha \models\!\!\dashv_\alpha (\models\!\!\dashv_\beta \beta)))$$

ki sta procesni izražavi, ko informacija β informira s svojima načinoma $\models_\beta$ ali $\models\!\!\dashv_\beta$ informacijska procesa $\models_\alpha \alpha$ ali $\alpha \models\!\!\dashv_\alpha$ ali ko je informacija α informirana z informacijskima procesoma $\beta \models_\beta$ ali $\models\!\!\dashv_\beta \beta$. To pa omogoča vpeljavo splošnejših ekvivalenc, ki sta

$$(\beta \models \alpha) \equiv ((\beta \models (\models \alpha)) \vee ((\beta \models) \models \alpha))$$
$$(\alpha \models\!\!\dashv \beta) \equiv (((\alpha \models\!\!\dashv) \models\!\!\dashv \beta) \vee (\alpha \models\!\!\dashv (\models\!\!\dashv \beta)))$$

S to posplošitvijo primera smo uporabili informacijski princip operatorske univerzalizacije, o kateri bo še govora.

Tako kot informira zunanja informacija, informira tudi protiinformacija. O tem specifičnem primeru informiranja bomo govorili v formulah, ki zadevajo protiinformacijo. S pravkar opravljeno razpravo pa smo zaenkrat izčrpali vsebino principa informacije. ■

Princip protiinformacije

Informacija prihaja v bivanje kot protiinformacija skladno s principom informacije. Protiinformacija informira kot informacija. Protiinformacija prihaja v bivanje kot razumevanje informacije z informacijo, kot regularna informacijska oblika ali regularni informacijski proces. Tudi protiinformacija je informacija sama po sebi. □

Formule protiinformacije

Protiinformacijo smo označili z ω, če pa je nastala iz informacije α pa še z ω_α ali $\omega(\alpha)$. Protiinformacija je informacija, torej nastaja spontano in cirkularno, tako da velja

$$\omega =_{Df} ((\omega \models) \vee (\models \omega)) \vee ((\models\!\!\dashv \omega) \vee (\omega \models\!\!\dashv)) \text{ in}$$
$$\omega =_{Df} ((\omega \models \omega) \vee (\omega \models\!\!\dashv \omega))$$

Vendar nastaja protiinformacija ω kot informacija v okviru informacije α, in sicer tako, da se iz α oziroma z vplivom α formira in da tudi sama formira informacijo α in seveda samo sebe. Informacijski sistem, ki je proces oblikovanja protiinformacije iz informacije in proces njenega vpliva na informacijo, je tedaj tale:

$$\alpha \models \omega;\ \omega \models \alpha;\ \alpha \models \alpha;\ \omega \models \omega;$$
$$\omega \models\!\!\dashv \alpha;\ \alpha \models\!\!\dashv \omega;\ \alpha \models\!\!\dashv \alpha;\ \omega \models\!\!\dashv \omega$$

Ta sistem je informacijska množica osmih procesov in zapišemo ga lahko v kompaktnejši obliki kot sistemsko formulo

$$\alpha, \omega \models \alpha, \omega;\quad \alpha, \omega \models\!\!\dashv \alpha, \omega$$

in preberemo takole: informacija α informira sebe in svojo protiinformacijo na en ali drug način in protiinformacija ω informira sebe in informacijo iz katere nastaja na en ali drug način.

Seveda pa je treba odgovoriti zadovoljivo še na vprašanje, ki je povezano s samim začetkom nastajanja protiinformacije ω. Pojav informacijskega začetka je mogoče opisati s posebnim operatorjem začetka pojavljanja protiinformacije na en (operator ⊔) ali drug način (operator ⊒). Tako imamo formulo

$$(\alpha \sqcup \omega) \vee (\omega \sqsupset \alpha)$$

Protiinformacija ω se začne pojavljati kot posledica informiranja informacije α na en ali drug način. Uporaba simetričnih operatorjev (npr. $\models$ in $\nvDash$, ⊔ in ⊒ itd.) ima še to dobro lastnost, da razen izražave raznovrstnosti informiranja omogoča še kompaktejši in nazornejši formulski zapis. Sedaj pa je mogoče razpravo o protiinformaciji strniti v naslednjo implikacijo:

$$((\alpha \sqcup \omega) \vee (\omega \sqsupset \alpha)) \Rightarrow (\alpha, \omega \models \alpha, \omega;\ \alpha, \omega \nvDash \alpha, \omega)$$

Če se je protiinformacija ω že pojavila, potem obstaja zapleteno medsebojno in lastno informiranje informacije α in protiinformacije na en ali drug način. ■

Doslej opisani principi nikakor niso zaprte izjave, saj so odvisni od povsem odprtih pojmov, ki jih bomo opredeljevali v naslednjih principih, posledicah in primerih. Razen tega bodo tudi kasnejši principi odvisni od prejšnjih, tako da bodo vsi rekurentno med seboj prepleteni in hkrati odprti za nadaljnji razvoj razumevanja.

Posledica informacijskih principov

Tudi sami informacijski principi bodo protiinformacijski. To pomeni, da bo neglede na informacijski princip v tem spisu, nek informacijski princip imel lastnost informacije, njenega cirkularnega in spontanega razvoja (mišljenja, nastajanja, vmeščanja) v carstvu informacije. □

To posledico potrjujejo tudi doslej zapisane informacijske formule, saj so v svoji pomenski zasnovi protiinformacijske, omogočajo ne le nastajanje novih formul, temveč tudi razvojno modifikacijo njih samih. Formalni sistem informacije ostaja vselej odprt in njegova formalna podoba je posledica trenutka oziroma dosežene stopnje formalnega razvoja. Formalni informacijski sistem ni zakonit matematičen sistem, vendar je matematičen sistem vselej le posebna, statična oblika formalnega informacijskega sistema.

Posledica informacijskega objekta in informacijskega subjekta

Informacija je informacijski objekt in/ali informacijski subjekt (obvladovanje same sebe). To ugotovitev je mogoče izraziti v formalizirani obliki, kjer je mogoče razumevati informacijo kot informacijski operand in/ali kot informacijski operator (operacijo). Informacijski objekti so tako informacijske oblike kot informacijski procesi, ki so hkrati informacijski operandi in informacijski operatorji. Informacijske objekte (operande) je mogoče formalno izraziti kot enote v oblikah, kot so informacijska_oblika, informacijski_proces, biti_v_obliki, biti_v_procesu itd. Podobno je mogoče informacijske subjekte (operatorje) izraziti z različnimi verbalnimi kompozicijami, kot so npr. informirati, generirati, nastajati, dajati, spreminjati, prihajati_v_eksistenco, prihajati_v_sedanjost, inteligentno_modificirati, vmeščati itd. □

Posledica protiinformacije

Protiinformacijski objekti in protiinformacijski subjekti so npr. protiinformacijska_oblika, protiinformacijski_proces, nastala_informacija itd. kot objekti in informirati_z_opazovanjem, generirati_s_spoznavanjem, nastajati_z_raziskovanjem kot subjekti itd. □

Princip informacijske formule

Informacijo je mogoče formalno izražati z informacijskimi in protiinformacijskimi formulami, ki so sestavljene iz informacijskih operandov (objektov) in informacijskih operatorjev (subjektov) v jezikovno svobodni, nevezani obliki. Informacijske operacije in njihove kompozicije se lahko uporabljajo kot informacijski operandi (objekti) in kot informacijski operatorji (subjekti). Čeprav je informacijska formula svobodno zaporedje (serijska izražava) informacijskih operatorjev in informacijskih operandov, je z njo (formalno-jezikovno) mogoče izražati cirkularnost, spontanost, paralelnost itd. oblik in procesov, ki se pojavljajo v informacijski formuli. □

V okviru te razprave je mogoče pokazati primere primitivnih in sestavljenih informacijskih formul. Princip informacijske formule omogoča naravno in formalno izražavo informacijskih oblik in procesov, uvajanje simbolov za informacijske operande in operatorje in naposled tudi formalistično (aksiomatsko, logično, matematično) oblikovanje informacijskih formul.

Primer osnovnega informacijskega sklepanja

Naslednja formula je deduktivno sklepanje, ki ima t.i. če-potem obliko oziroma je implikacija:

(informacija informira
lastno_in_drugo_informiranje;
informiranje informira informacijo;)

---

informacija informira informacijo;

Posamezne izjave se končujejo s podpičjem. Seznam izjav lahko predstavlja vobče izjavo, ki povezuje dele seznama z vezmi tipa "in", "ali" ali z "in/ali" itd. □

Formula enostavnega informacijskega sklepanja

Zadnji primer je mogoče zapisati enostavno tudi v formalni obliki. Informiranje informacije α označimo z $\Im_\alpha$, informiranje druge informa-

cije β pa z $\mathfrak{I}_\beta$. Črta pod formulama pomeni informacijski operator ekstracije. Podobno črtno operacijo uporabljamo npr. pri sklepanju po načelu modus ponens. Formulo za zadnji primer lahko zapišemo takole:

$$\frac{((\alpha \models \mathfrak{I}_\alpha, \mathfrak{I}_\beta);\ (\mathfrak{I}_\alpha, \mathfrak{I}_\beta \models \alpha);\ (\mathfrak{I}_\alpha, \mathfrak{I}_\beta \models \alpha);\ ((\alpha \models \mathfrak{I}_\alpha, \mathfrak{I}_\beta))}{\alpha \models \alpha;\ \alpha \models \alpha}$$

Ta zapis dodaja zapisu iz zadnjega primera še informiranje na en ali drug način. Takšno obliko sklepanja v obliki formule bomo imenovali modus informationis. Ti modusi bodo vselej povezani z nastajalnim ekstrahiranjem informacije iz danih informacijskih agregatov. Zadnjo formulo lahko zapišemo tudi tako, da zamenjamo vodoravno črto (operacijo ekstrakcije) z informacijskim operatorjem "/", ki je ustrezna partikularizacicija metaoperatorja $\models$. ■

Primer informacijske ekvivalence

Tale formula opisuje ekvivalenco:

informacija informira inteligenco
je_enako_kot
informacija informira informiranje;
informiranje informira inteligenco;

Operande in operatorje formule je mogoče konstruirati s svobodno izbiro. Izjava pred operatorjem se ne končuje s posebnim znakom. □

Primer formule za informacijsko ekvivalenco

Zadnji primer lahko opišemo formalno z uvedbo naslednjih oznak: α označuje informacijo, $\mathfrak{I}$ inteligenco in $\mathfrak{I}_\alpha$ informiranje; = je znak za operator "je_enako_kot". Dobimo:

$$((\alpha \models \mathfrak{I}) \vee (\mathfrak{I} \models \alpha)) = (((\alpha \models \mathfrak{I}_\alpha) \wedge (\mathfrak{I}_\alpha \models \mathfrak{I})) \vee ((\mathfrak{I}_\alpha \models \alpha) \wedge ((\mathfrak{I} \models \mathfrak{I}_\alpha)))$$

Operator $\wedge$ označuje povezavo "in". Ta formula seveda ni kak končen dosežek, saj smo v njen izpustili vpliv inteligence na informiranje informacije. To konstrukcijo pa si lahko bralec po tem, kar je bilo povedanega, naredi že sam.

Na mestu pa je še ena pripomba. Od nekako še vedno informacijsko statičnih konstrukcij, ki so bržkone posledica matematične tradicije simbolizma in njegovega dojemanja, je mogoče tudi pri izražavi formul ubrati t.i. dinamično interpretacijo oziroma tudi izražavo. Zapišimo zadnji primer še v bolj živi obliki! ■

Primer dveh smiselnih formul

Tile dve formuli sta smiselni: prva opisuje operacijsko kompozicijo, druga pa ekvivalenco dveh formul:

(1) informirati
je_enako_kot
informirati, generirati, modificirati;

(2) informacija informira inteligenco
je_enako_kot
informacija informira, generira, modificira
inteligenco
je_enako_kot
informacija informira inteligenco,
informacija generira inteligenco,
informacija modificira inteligenco;

V tem primeru se nekatere izjave končujejo z vejico, ki predstavlja posebno "in" povezavo itd. □

Formalizacija dveh smiselnih formul

Zadnja dva primera lahko zapišemo v simbolični obliki takole:

(1) $\mathfrak{I} = \mathfrak{I}, \mathfrak{I}_{gen}, \mathfrak{I}_{mod}$

To je informacijska formula s partikulariziranima implicitnima oparatorjema $\mathfrak{I}_{gen}$ in $\mathfrak{I}_{mod}$, ki imata zaenkrat nedoločene argumente. Entiteto $\mathfrak{I}$ in njeni partikularizaciji je mogoče razumeti kot informiranje. Informacijski operator = je v bistvu partikularizacija operatorja $\models$ (konkretno bi lahko zapisali $\models_=$). Zaporedje $\mathfrak{I}$, $\mathfrak{I}_{gen}$, $\mathfrak{I}_{mod}$ je informacijska množica na desni strani operatorja =. Formula je sama na sebi rekurzivna. Njena logična dekompozicija bi bila tale:

$$\mathfrak{I} = \mathfrak{I};\ \mathfrak{I} = \mathfrak{I}_{gen};\ \mathfrak{I} = \mathfrak{I}_{mod}$$

Lahko pa bi osnovni informacijski zapis pojasnili tudi v obliki logično ortodoksne formule:

$$(\mathfrak{I} = \mathfrak{I}, \mathfrak{I}_{gen}, \mathfrak{I}_{mod}) \equiv ((\mathfrak{I} = \mathfrak{I}) \wedge (\mathfrak{I} = \mathfrak{I}_{gen}) \wedge (\mathfrak{I} = \mathfrak{I}_{mod}))$$

(2) V tej formuli označimo z Š inteligenco kot informacijo. Tako lahko zapišemo:

$$(\alpha \models \mathfrak{I}) = ((\alpha \models \circ \models_{gen} \circ \models_{mod} \mathfrak{I}) = ((\alpha \models \mathfrak{I}) \wedge (\alpha \models_{gen} \mathfrak{I}) \wedge (\alpha \models_{mod} \mathfrak{I})))$$

Ta formula je med drugim tudi primer možnosti dinamičnega informiranja, saj se iz prvega člena glavne ekvivalence na desni strani informira ekvivalentno drugi člen kot konjunkcija treh členov. V tem primeru smo tudi pokazali, kako je mogoče oblikovati operatorske kompozicije, ki bi bile lahko posledica verbalnih oblik izražanja. Takšna kompozicija je "operatorski" izraz $\models \circ \models_{gen} \circ \models_{mod}$. Seveda bi bilo mogoče operatorske kompozicije obravnavati kot posebne oblike informacijskih formul, npr. kot informacijske podformule. ■

Posledica oblike operatorja "informirati"

Najsplošnejši (in morda najmočnejši) informacijski operator je operator "informirati". Ta operator, ki je metaoperator za poljubni operator, se razumeva kot kompozicija poljubnega zaporedja drugih informacijskih operatorjev. Vsaka operatorska kompozicija lahko ima pomen njihove čiste paralelnosti, mešane paralelnosti in serialnosti ali čiste serialnosti. Vrstni red posameznega operatorja v sestavljenem zapo-

redju je lahko ali pa tudi ne določen glede na posamezne formalne povezave. Informacijski metaoperator je mogoče vselej partikularizirati ali univerzalizirati skladno s potrebami, uporabo, ali zahtevo. Informacijski metaoperator v formuli se vede kot informacijska spremenljivka. □

Formalni pomen in oblika operatorja "informirati"

Z operatorjema $\models$ in $\dashv$ smo označevali kompleksnost informiranja informacije na en ali drug način. Ta metaoperatorja smo uporabljali kot spremenljivki, ki bosta dobili konkretno funkcijo v konkretnem primeru. Izza njiju smo "skrili" še princip partikularizacije in univerzalizacije, in seveda t.i. možnost kompozicioniranja. S tem smo metaoperatorjema podelili kar največjo moč spontanosti in cirkularnosti.

Praktično vprašanje, ki se ob vsem tem lahko pojavi, je, kako bi bilo mogoče nek spontan informacijski operator realizirati. Ali je to sploh mogoče? Odgovor na to vprašanje bomo dali v epilogu tega spisa, ko bodo bolj znani še drugi informacijski pojavi oziroma njihovi principi. ■

PRINCIPI INFORMIRANJA INFORMACIJE

V tem poglavju bomo preučevali več principov, ki izhajajo iz pomena glagola "informirati" in nekatere odvode teh principov. Naj glagol "informirati" vključuje pomene vseh mogočih glagolov in verbalnih kompozicij. V tem primeru lahko predstavlja glagol "informirati" poljubno zapleten operator. Naj bo ta pomen transparenten tudi za samostalnik "informacija". Tako je informacija lahko razumevana kot poljubna oblika, poljuben proces ali oboje. Tako lahko predstavlja informacija poljubno zapleten informacijski operand (objekt) in informacijski operator (subjekt).

Informiranje informacije zadeva kot možno razumevanje štiri osnovne oblike ali procese informacije, ki so: informiranje, vmeščanje, nastajanje in protiinformiranje informacije. Ta refleksija daje naslednje štiri principe.

Princip informiranja informacije

Informiranje informacije, kjer je informiranje inherentno sami informaciji, pomeni, da informacija informira z informiranjem skladno s principom informacije, da prihaja z informiranjem v eksistenco protiinformacija skladno s principom protiinformacije in da se protiinformacija vmešča v obstoječo informacijo skladno s principom vmeščanja informacije. Informiranje cirkularno in spontano obvladuje informacijo skladno s principi cirkularnosti, rekurence, paralelnosti in serialnosti informacije (glej naslednje principe). Informiranje informacije je tudi samo informacija. □

Formula informiranja informacije

Prejšnji princip našteva tale dejstva: (1) informiranje $\Im(\alpha)$ je inherentno informaciji α; pojavlja se torej proces

$$(\alpha \models \Im(\alpha)) \vee (\Im(\alpha) \models \alpha) \vee (\Im(\alpha) \dashv \alpha) \vee (\alpha \dashv \Im(\alpha))$$

ali splošeje inherenten (tudi notranji) proces

$$(c \models_{\Im(\alpha)}) \vee (\models_{\Im(\alpha)} \alpha) \vee (\dashv_{\Im(\alpha)} \alpha) \vee (\alpha \dashv_{\Im(\alpha)})$$

(2) Z informiranjem $\Im$ prihaja v obstajanje protiinformacija ω:

$$(\Im(\alpha) \llcorner \omega) \vee (\omega \lrcorner \Im(\alpha)) \vee (\Im(\alpha) \models \omega) \vee (\omega \dashv \Im(\alpha))$$

Za ta primer je mogoče uporabiti tudi operatorski kompoziciji, in sicer:

$$(\Im(\alpha) \llcorner\circ\models \omega) \vee (\omega \dashv\circ\lrcorner \Im(\alpha))$$

Vendar je mogoče tudi skladno z definicijo operatorja $\models$ in operatorja $\dashv$ predpostaviti, da je pojavitev informacijskega v njima že vsebovana, torej $(\llcorner) \subset (\models)$ in $(\lrcorner) \subset (\dashv)$. (3) Nastala protiinformacija se vmešča v informacijo. Ta proces je mogoče izraziti dinamično s procesoma

$$((\Im(\alpha) \llcorner\circ\models \omega) \vee (\omega \dashv\circ\lrcorner \Im(\alpha))) \models_{\mathfrak{E}(\alpha)} \alpha;$$
$$\alpha \dashv_{\mathfrak{E}(\alpha)} ((\Im(\alpha) \llcorner\circ\models \omega) \vee (\omega \dashv\circ\lrcorner \Im(\alpha)))$$

Ta primer formalizacije kaže, kako je mogoče posamezne primere izračati vedno bolj zapleteno do potrebnih podrobnosti. (4) Informiranje informacije je tudi paralelno in cirkularno (v pomenu ciklično). Tu je potrebno poudariti, da sta operatorja $\models$ in $\dashv$ tudi predstavnika splošno paralelnih ($\|\models$, $\|\dashv$), cikličnih ali cirkularnih ($\circlearrowright\models$, $\circlearrowright\dashv$) in naposled tudi paralelno cikličnih operatorjev ($\|\circlearrowright\models$, $\|\circlearrowright\dashv$). (6) Informiranje informacije je tudi samo informacija: to lastnost je mogoče enostavno opisati s formulo $\Im(\alpha) \subset \alpha$. ■

Princip vmeščanja informacije

Vmeščanje informacije pomeni povezovanje, vgnezdevanje ali depozicijo (tudi interpretacijo, kodiranje, pomensko prepletanje, razumevanje itd.) svobodne, nastajajoče, protiinformacijske informacije v dano pojmovno (pojmovnostno, komprehenzivno) telo ali domeno informacije. Z vmeščanjem se informacija postavlja v regularni informacijski kontekst (v možne informacijske relacije), v območje obstoječe informacije (z možnim pomenom). Na jezikovni ravni se pojavlja semantična relacija med novo in obstoječo informacijo. Vmeščanje informacije je tudi samo informacija. □

Formule vmeščanja informacije

Naj bo β svobodna informacija, ω protiinformacija in α informacija, v katero se β in ω vmeščata. Vmeščanje $\mathfrak{E}$ pomeni, da β in ω informirata α. Torej imamo

$$(\beta, \omega \models_{\mathfrak{E}} \alpha) \vee (\alpha \dashv_{\mathfrak{E}} \beta, \omega)$$

namesto splošnejšega

$$(\beta, \omega \models \alpha) \vee (\alpha \dashv \beta, \omega)$$

Na tem mestu se pojavi bistvena dilema: če vmeščanje $\mathfrak{E}$ obstaja in β in ω dejansko vplivata na α (dejansko informirata α), potem operatorja vmeščanja $\models_{\mathfrak{E}}$ in $=\!|_{\mathfrak{E}}$ za samo vmeščanje nista operatorski kompoziciji oblik $\models_\beta \circ \not\models_{\mathfrak{E}(\alpha)}$, $\models_\omega \circ \not\models_{\mathfrak{E}(\alpha)}$, $\not=\!|_{\mathfrak{E}(\alpha)} \circ =\!|_\beta$ in $\not=\!|_{\mathfrak{E}(\alpha)} \circ =\!|_\omega$. V teh kompozicijah se namreč pojavljata operatorja nevmeščevalnega informiranja $\not\models_{\mathfrak{E}(\alpha)}$ in $\not=\!|_{\mathfrak{E}(\alpha)}$ glede na α. Nezmožnost vmeščanja svobodne informacije β in protiinformacije ω v informacijo α na en ali drug način bi tedaj bilo mogoče izraziti s sistemom formul

$$\beta \models_\beta \circ \not\models_{\mathfrak{E}(\alpha)} \alpha;\ \omega \models_\omega \circ \not\models_{\mathfrak{E}(\alpha)} \alpha;$$
$$\alpha \not=\!|_{\mathfrak{E}(\alpha)} \circ =\!|_\beta \beta;\ \alpha \not=\!|_{\mathfrak{E}(\alpha)} \circ =\!|_\omega \omega$$

V tem primeru bi rekli, da sta informacija β in ω informacijski šum za informacijo α. V tem primeru bi lahko tudi postavili

$$\mathfrak{E}(\alpha) \subset \alpha, \text{ vendar } \neg\mathfrak{E}(\alpha) \not\subset \alpha$$

V zadnji formuli označuje $\neg\mathfrak{E}(\alpha)$ nevmeščanje glede na α. Tako bi namesto prejšnjih operatorskih kompozicij lahko postavili tudi kompozicije nevmeščanja $\models_\beta \circ \models_{\neg\mathfrak{E}(\alpha)}$, $\models_\omega \circ \models_{\neg\mathfrak{E}(\alpha)}$, $=\!|_{\neg\mathfrak{E}(\alpha)} \circ =\!|_\beta$ in $=\!|_{\neg\mathfrak{E}(\alpha)} \circ =\!|_\omega$. ■

Princip nastajanja informacije

Nastajanje informacije je generiranje, razvijanje, izginjanje in prihajanje informacije v eksistenco. Nastajanje informacije ima moč novega informacijskega pojavljanja, prihajanja nove informacije v prezenco. Nastajanje informacije ni informacijsko omejeno, je cirkularno in spontano. Z nastajanjem se lahko pojavi katera koli informacija. Nastajanje informacije je tudi samo informacija. □

Formule nastajanja informacije

Nastajanje informacije α je formulsko določeno z $(\alpha \models) \vee (\models \alpha) \vee (=\!| \alpha) \vee (\alpha =\!|)$, ko informacija nastaja ba en ali drug način. Z informacijo α nastaja nova informacija, imenovana protiinformacija ω, torej vobče $(\alpha \sqcup \omega) \vee (\omega \sqcup \alpha) \vee (\alpha \models \omega) \vee (\omega =\!| \alpha)$. Pri tem velja za protiinformacijo seveda $(\omega \models) \vee (\models \omega) \vee (=\!| \omega) \vee (\omega =\!|)$. Nastajanje informacije α je tudi samo informacija, torej

$$((\alpha \models) \vee (\models \alpha) \vee (=\!| \alpha) \vee (\alpha =\!|)) \subset \alpha$$

Nastajanje ali informiranje $\mathfrak{I}$ informacije α je mogoče smiselno označiti z $\mathfrak{I}(\alpha)$, torej lahko zapišemo namesto prejšnjega $\mathfrak{I}(\alpha) \subset \alpha$. ■

Princip protiinformiranja informacije

Protiinformiranje informacije pomeni, da je informacija s svojim informiranjem dojemljivostno (obveščevalno, raziskovalno in/ali spoznavno) informirana. Produkt tega informiranja je protiinformacija. Protiinformiranje je produkcija opazovalne, raziskovalne in spoznavne informacije v procesu informiranja. Protiinformiranje je tudi samo informacija. □

Formule protiinformiranja informacije

S pojavljanjem protiinformacije je mogoče povezati tudi tisti del informiranja informacije α, ki povzroča pojavljanje in nastajanje protiinformacije ω. To komponento informiranja $\mathfrak{I}(\alpha)$ imenujemo protiinformiranje in ga označimo s $\mathfrak{C}$ ali $\mathfrak{C}(\alpha)$. Informacija α proizvaja svoje informiranje $\mathfrak{I}$ in v okviru tega tudi skupaj z $\mathfrak{I}$ svoje protiinformiranje $\mathfrak{C}$, ki proizvaja protiinformacijo ω. Tako je mogoče formalno izraziti s poldinamičnim sistemom:

$$((\alpha \models \mathfrak{I}(\alpha)) \vee (\mathfrak{I}(\alpha) =\!| \alpha) \models \mathfrak{C}(\alpha)) \models \omega;$$
$$\omega =\!| ((\alpha \models \mathfrak{I}(\alpha)) \vee (\mathfrak{I}(\alpha) =\!| \alpha) \models \mathfrak{C}(\alpha))$$

Na tem primeru vidimo, kako je protiinformiranje $\mathfrak{C}$ dejansko implicitna komponenta informacije α in njenega informiranja $\mathfrak{I}$. Protiinformiranje $\mathfrak{C}$ je posledica opazovanja $\mathfrak{C}_{obs}$, raziskovanja $\mathfrak{C}_{inv}$ in spoznavanja $\mathfrak{C}_{cog}$ informacije α. To so tri bistvene komponente protiinformiranja, ki vsaka za sebe prispevajo k procesu pojavljanja in nastajanja protiinformacije. V okviru te razprave je mogoče konstruirati še vrsto formul, ki zadevajo protiinformiranje. Naposled je tudi protiinformiranje $\mathfrak{C}(\alpha)$ samo po sebi informacija, torej informira kot informacija in je podoblika oziroma podproces zadevne informacije α: $\mathfrak{C}(\alpha) \subset \alpha$. ■

Posledica informiranja

Informiranje informacije je oblika in/ali proces cirkularnega in spontanega informiranja, vmeščanja, nastajanja in protiinformiranja informacije. □

Formule informiranja

Informiranje $\mathfrak{I}(\alpha)$ informacije α je oblika in/ali proces cirkularnega in spontanega informiranja $\mathfrak{I}(\alpha)$, protiinformiranja $\mathfrak{C}(\alpha)$ in vmeščanja $\mathfrak{E}(\alpha)$ informacije α in druge informacije β. Informiranje je nastajalno in se eksplicitno izraža z operatorji tipa $\models$ in/ali $=\!|$. Formule informiranja so tedaj vse možne formule, ki so kombinacija operandov tipa α, implicitnih operatorjev tipa $\mathfrak{I}(\alpha)$, partikulariziranih eksplicitnih operatorjev tipa $\models_{\mathfrak{I}(\alpha)}$ in tipa $=\!|_{\mathfrak{I}(\alpha)}$ in univerzaliziranih eksplicitnih operatorjev tipa $\models$ in $=\!|$. Vsi elementi informacijske formule so ali operatorji ali operandi. To velja tudi za oklepaje, ki so ločilni operatorji, npr. partikularizacija tipa $\models_($ in $=\!|_)$. Tudi operatorske kompozicije niso kaj drugega kot informacijsko sestavljeni operatorji. ■

Primer informiranja

Zadnjo posledico je mogoče izraziti formalizirano takole:

informiranje
pomeni
informiranje, vmeščanje, nastajanje,
protiinformiranje;

Ta formula je rekurzivna v informiranju, vmeščanju, nastajanju in protiinformiranju. Vse te entitete so informacijsko prepletene. □

Primer formule informiranja

Zadnji verbalni primer je mogoče formalno izraziti na več načinov. Oglejmo si le dva možna primera! V prvem primeru imamo:

$$\mathfrak{I} = \mathfrak{I}, \mathfrak{E}, \mathfrak{I}_{nas}, \mathfrak{C}$$

To je navadna informacijska formula s partikulariziranim eksplicitnim operatorjem, ki je $\models_{=}$. V drugem primeru naj bosta dani še informacija α in β. Tako je mogoče konstruirati primer

$$(\alpha \models_{\mathfrak{I}} \beta) \models_{=} ((\alpha \models_{\mathfrak{I}} \beta) \vee (\alpha \models_{\mathfrak{E}} \beta) \vee (\alpha \models_{\mathfrak{I}(nas)} \beta) \vee (\alpha \models_{\mathfrak{C}} \beta)) \blacksquare$$

Primer informacijskega vmeščanja

Princip informacijskega vmeščanja je mogoče interpretirati s tole formulo:

vmeščanje
pomeni
ali
je_enako_kot
semantična_povezava
protiinformacije
in
obstoječe_informacije;

kjer je vsaka entiteta zapisana v posebni vrstici. Metaforična predstavitev te formule poudarja pomen in druge možnosti formulskega izražanja. Tako je npr. zapis te formule v poljski (prefiksni) obliki daljši:

ali pomeni
vmeščanje semantična_povezava in
protiinformacije
obstoječe_informacije
je_enako_kot
vmeščanje semantična_povezava in
protiinformacije
obstoječe_informacije;

V tem primeru je mogoče binarne (ali večkratne) operatorje "ali", "pomeni", "in", "je_enako_kot" razumevati kot paralelne operacije, funkcijo "semantična_povezava" kot unarni operator, "protiinformacija" in "obstoječa_informacija" kot paralelna procesa in "vmeščanje" kot paralelni, dvodelni rezultat. □

Formule informacijskega vmeščanja

Naj označuje $\mathfrak{E}$ vmeščanje, $=_{pom}$ operator "pomeni", $\vee$ operator "ali", $=$ operator "je_enako_kot", $\mathfrak{S}$ semantično_povezavo, ω protiinformacijo in α obstoječo informacijo. Prva formula iz prejšnjega primera ima tedaj formalno obliko

(1) $$((\mathfrak{E} =_{pom}) \vee (\mathfrak{E} =)) \models \mathfrak{S}(\omega, \alpha)$$

Operator $\models$ ima pomen "informira" ali "je_res". Drugi primer pa je prefiksni poljski zapis te formule, in sicer v neregularni fomulski obliki

(2) $$\vee =_{pom} \mathfrak{E}\ \mathfrak{S}\ \omega\ \alpha = \mathfrak{E}\ \mathfrak{S}\ \omega\ \alpha$$

To formulo je mogoče zapisati v regularni, še vedno v pretežno prefiksni obliki takole:

(3) $$\models_{\vee} (=_{pom} \mathfrak{E}, \mathfrak{S}(\omega, \alpha)), (= \mathfrak{E}, \mathfrak{S}(\omega, \alpha))$$

Pri tem je mogoče $\mathfrak{S}(\omega, \alpha)$ eksplicirati z nazornejšo formulo

$$(\omega, \alpha \models_{\mathfrak{S}} \omega, \alpha) \vee (\omega, \alpha \not\models_{\mathfrak{S}} \omega, \alpha)$$

Operator $\models_{\vee}$ beremo npr. informira se ena ali druga ali obe možnosti (informacijski entiteti), ki sledita temu operatorju v formuli (3). Nadalje preberemo npr. $=_{pom}$ kot informira se enak pomen za operanda, ki temu operatorju sledita. In temu podobno. ■

Primeri informacijskega nastajanja

Primeri formul z informacijskim nastajanjem so tile:

(1) Formula informacijskega deduktivnega sklepanja (implikacija) je:

informacija generira protiinformacijo;
protiinformacija se_vmešča_v informacijo;
---
protiinformacija postaja regularna_informacija;

(2)
nastajati
ima_pomen
biti, biti_v_kontradikciji, biti_v_protiakciji,
delati, generirati, informirati, inovirati,
izvirati, izzivati, kreirati, mutirati,
na_novo_procesirati, narediti, nasprotovati,
nastajati, navdihovati, oblikovati, odkrivati,
omogočati, pojavljati, postavljati,
predstavljati, prihajati_na_dan,
prihajati_v_bivanje, prihajati_v_eksistenco,
prihajati_v_prezenco, prihajati_v_razumevanje,
producirati, protiinformirati, rasti,
razvijati, uvajati, uveljavljati, vzkliti,
vznemirjati, ... ;

(3) informacija generira informacijo
je_enako_kot
informacija je_generirana_iz informacije; □

Formule za primere informacijskega nastajanja

(1) Deduktivno sklepanje ali tudi modus ponens se lahko uvrščajo v okvir splošnejšega informacijskega transformacijskega principa, ki ga imenujemo modus informationis. Za prejšnji primer je mogoče konstruirati tole pravilo:

$$\frac{(\alpha \models \omega) \vee (\omega \not\models \alpha);\ (\omega \models_{\mathfrak{E}} \alpha) \vee (\alpha \not\models_{\mathfrak{E}} \omega)}{\omega \therefore_{\mathfrak{C}} \alpha}$$

Antecedens tega pravila predvideva generiranje protiinformacije ω iz informacije α na en ($\models$) ali drug način ($\not\models$) in tudi vmeščevanje nastale protiinformacije na en ($\models_{\mathfrak{E}}$) ali drug način ($\not\models_{\mathfrak{E}}$). Konsekvens pravila upošteva približno

(ali podobnostno) vključenost protiinformacije ω v informacijo α. Relacija podobnostne vključenosti je simbolizirana z operatorsko kompozicijo $\therefore\circ C$. Podrobna razlaga te kompozicije je prepuščena individualnemu okusu.

(2) Glagol nastajati, ki ga reprezentirata informacijska metaoperatorja $\models$ in $\dashv$, ima lahko vrsto partikularnih glagolskih pomenov oziroma pomen vseh partikularnih pomenov, ki niso samo glagolski, temveč so poljubno informacijsko kompleksni (univerzalizirani).

(3) Ta primer upošteva različne možnosti branja (ali pomena) formule $(\alpha \models \alpha) \vee (\alpha \dashv \alpha)$. Mogoče je prebrati tole:

$$(\alpha \models \alpha) = (\alpha \dashv \alpha)$$

Informacija generira informacijo je enako kot informacija je generirana iz informacije. ■

Primer formule s protiinformiranjem

Protiinformiranje ima kot informacija svoj izvor v informiranju informacije. Protiinformiranje je pojmovnostni del informiranja, ki opazuje, raziskuje in razpoznava informacijo in informiranje informacije. Protiinformiranje se pojavlja med informiranjem informacije kot informacijski protipojav, ki iz informiranja izrašča ali je generirano z informiranjem. Tako je mogoče zapisati tole informacijsko formulo:

protiinformiranje
izvira_iz ali je_generirano_z
informiranjem_informacije
kot
opazovanje, raziskovanje, razpoznavanje
informacije, informiranja informacije;

Predzadnja in zadnja vrstica formule sta v razmerju operatorjev in operandov (v angleščini je med njima predlog "of"). □

Formula za primer s protiinformiranjem

Zapišimo formalno prejšnji primer takole:

$$(\mathfrak{I}(\alpha) \dashv \alpha) \Rightarrow (\mathfrak{C}(\alpha) \dashv \mathfrak{I}(\alpha))$$

Ta primer preberemo: če je informiranje informacije $\mathfrak{I}(\alpha)$, ki je opazovanje, raziskovanje in razpoznavanje informacije α in informiranja informacije (po definiciji pojma informacije), generirano (operator $\dashv$) z informacijo, potem protiinformiranje $\mathfrak{C}(\alpha)$ izvira ali je generirano (operator $\dashv$) z informiranjem informacije. ■

Seveda je mogoče zapisati še več drugih formul za protiinformiranje tudi v bolj podrobni obliki.

PRINCIPI INFORMACIJSKE CIRKULARNOSTI

Kako nastaja informacija cirkularno? Kakšna je narava informacijske cirkularnosti? Kakšen je vpliv informacijske cirkularnosti na carstvo informacije? Nekaj odgovorov na ta vprašanja bo danih s principi, posledicami in primeri v naslednjih vrsticah.

Princip cirkularnosti informacije

Informacija je spontano cirkularna glede na informacijo. Informacijska cirkularnost je lastnost celotne informacijske domene (protiinformacije, informiranja, vmeščevanja, nastajanja, protiinformiranja) in je informacijsko transparentna. Različne oblike in procesi informacijske cirkularnosti so rekurenca, paralelnost, serialnost (posledičnost) pa tudi neidentificirane oblike in procesi informacije. Cirkularnost informacije je informacija sama po sebi. □

Formule cirkularnosti informacije

Informacija α je spontano cirkularna glede na samo sebe. Ta cirkularnost se izraža tudi s sistemsko (ne z disjunktivno) definicijo informacije

$$\alpha =_{Df} (\alpha \models; \models \alpha; \dashv \alpha; \alpha \dashv)$$

ki pravi, da informacija α odprto informira in/ali je odprto informirana na en ali drug spontan način. Spontanost je zajeta v naravi operatorjev $\models$ in $\dashv$ in s tem v sami informaciji α. Iz definicije informacije, tj. iz njene cirkularnosti, je mogoče izpeljati tudi formule

$\alpha \models \alpha;$
$\alpha \models (\alpha \models); \alpha \models (\models \alpha); (\alpha \models) \models \alpha; (\models \alpha) \models \alpha;$
$\alpha \dashv \alpha;$
$(\dashv \alpha) \dashv \alpha; (\alpha \dashv) \dashv \alpha; \alpha \dashv (\dashv \alpha); \alpha \dashv (\alpha \dashv);$
$(\alpha \models \alpha) \models (\alpha \dashv \alpha); \ldots$

Formule je tedaj mogoče izpeljevati do poljubnih rekurzivnih globin. Zaradi informacijske transparentnosti se spontana cirkularnost prenaša na poljubno informacijo, torej tudi na protiinformacijo ω, informiranje $\mathfrak{I}$, vmeščanje $\mathfrak{E}$ in protiinformiranje $\mathfrak{C}$. Tako imamo npr.

$\omega \models \omega; \alpha \models (\omega \models); \omega \models (\dashv \alpha); (\models \mathfrak{C}) \dashv \alpha;$
$\omega \models (\mathfrak{C} \models (\mathfrak{E} \models (\mathfrak{I} \models \alpha))); \ldots$

Če se s formulo $\alpha \models \alpha$ izraža rekurenca, se t.i. splošna paralelnost informacijskih procesov opisuje z operatorjema $\Vdash$ in $\dashv\!\!\dashv$ in njunimi partikularizacijami in univeralizacijami. Z njima (ali prek njiju) se informacijski procesi paralelno in medsebojno informacijsko prepletejo, kot kaže primer paralelnega cikličnega informiranja informacije:

$\alpha \Vdash \mathfrak{I}; \mathfrak{I} \dashv\!\!\dashv \alpha; \mathfrak{I} \Vdash \alpha; \alpha \dashv\!\!\dashv \mathfrak{I};$
$\alpha, \mathfrak{I} \Vdash \mathfrak{C}; \mathfrak{C} \dashv\!\!\dashv \alpha, \mathfrak{I}; \alpha, \mathfrak{I}, \mathfrak{C} \Vdash \omega; \alpha \dashv\!\!\dashv \alpha, \mathfrak{I}, \mathfrak{C};$
$\alpha, \mathfrak{I}, \mathfrak{C}, \omega \Vdash \mathfrak{E}; \mathfrak{E} \dashv\!\!\dashv \alpha, \mathfrak{I}, \mathfrak{C}, \omega;$
$\omega, \mathfrak{E} \Vdash \alpha, \mathfrak{I}; \alpha, \mathfrak{I} \dashv\!\!\dashv \omega, \mathfrak{E}; \ldots$

Posebna oblika cirkularnosti informacije je t.i. ciklično informiranje, ki pomeni, da informacija nastaja s protiinformacijo in se protiinformacija vmešča v obstoječo informacijo. Cikličnost informacije smo med drugim opisali prav z zadnjim sistemom formul. Za označevanje informacijske cikličnosti v opisanem in splošnem pomenu uporabimo informacijska operatorja $\vdash$

in $\dashv$. Tako ima $\alpha \vdash \alpha$ pomen, da imamo v tem procesu vsaj en cikličen proces. V procesih, kjer želimo poudariti njihovo specifično necikličnost, bi lahko uporabili operatorja $\nvdash$ in $\nvdash$ oziroma njune partikularizacije. ■

Princip informacijske rekurence

Informacijska rekurenca je informacijsko inherentna in informacijsko transparentna lastnost cirkularnosti informacije, ki je fenomenološka, metodološka, rekurzivno formalna, lingvistična, biološka itd. Ta lastnost je spontana, spoznavajoča, informacijsko generativna, paralelna, serijska, posledična itn. Informacijska rekurenca je informacija tudi sama. □

Formule informacijske rekurence

Formule informacijske rekurence opisujejo informacijsko cirkularnost in informacijsko cikličnost, kot specifično, protiinformacijsko/vmeščevalno cirkularnost. Splošni, paralelni, ciklični in paralelno ciklični procesi rekurence ne en ali drug način so npr. tile:

$$\alpha \models \alpha;\ \alpha \dashv\models \alpha; \qquad \alpha \models\!\models \alpha;\ \alpha \dashv\models\!\models \alpha;$$
$$\alpha \vdash \alpha;\ \alpha \dashv \alpha; \qquad \alpha \Vdash \alpha;\ \alpha \dashv\Vdash \alpha$$

Paralelno ciklična rekurenca ($\Vdash$, $\dashv\Vdash$) je vobče paralelno in serijsko prepletena informacija. Metaoperatorja $\models$ in $\dashv\models$ sta tudi najbolj splošna, tako da so ostali metaoperatorji ($\models\!\models$, $\dashv\models\!\models$, $\vdash$, $\dashv$, $\Vdash$, $\dashv\Vdash$) že njune partikularizacije. Podobno je mogoče operator $\models$ uporabljati kot najsplošnejši metaoperator, ki lahko predstavlja tudi vse operatorje tipa $\dashv\models$ oziroma njegove partikularizacije.

Seveda je mogoče eksplicitno izraziti tudi univerzalizirano ali partikularizirano lastnost informacijske nerekurence. Tako imamo primere splošne, paralelne, ciklične in paralelno ciklične informacijske nerekurence:

$$\alpha \not\models \alpha;\ \alpha \not\dashv\models \alpha; \qquad \alpha \not\models\!\models \alpha;\ \alpha \not\dashv\models\!\models \alpha;$$
$$\alpha \nvdash \alpha;\ \alpha \not\dashv \alpha; \qquad \alpha \nVdash \alpha;\ \alpha \not\dashv\Vdash \alpha$$ ■

Posledica informacijske rekurence

Beseda "rekurenca" ima pomen vračanja informacije v eksistenco, vmeščevanje, informiranje in protiinformiranje, in sicer v spontanem krogu, ki ga razumevamo kot informiranje v najširšem pomenu. Primer statične, konstruktivne informacijske rekurence je matematična rekurzivna formula, ki omogoča rekurzivno pridobivanje novih formalnih ali numeričnih (logičnih) rezultatov z uporabo formalnih lingvističnih pripomočkov. Informacijska rekurenca je dinamična, tako da se npr. omenjena rekurzivna formula spreminja in razvija z rekurenčnim procesom. Razen tega je potrebno informacijsko rekurenco razumevati kot informacijsko paralelno in/ali informacijsko serijsko (posledično) rekurenco. Izgleda, kot da je serijska rekurenca očitna in predstavljiva, čeprav je potrebno upoštevati možno nastajanje informacije v okviru serijsko strukturirane informacije. Toda paralelna rekurenca mora upoštevati možnosti nastajanja paralelne informacije v okviru že obstoječe paralelne in serijske informacije. V tem smislu informacijska rekurenca ni nikjer omejevana in informacija in njeno informiranje lahko spontano nastajata na informacijsko cirkularen način. Končno je smiselno poudariti, da upošteva informacija protiinformacijsko tudi informacijsko rekurenco kot informacijsko cirkularnost s svojim notranjim informacijskim dojemanjem, ki je lastnost same informacije. □

Primer splošne informacijske rekurence

Informacijsko rekurenco, v kateri paralelnost in/ali serialnost informacijskih oblik in/ali informacijskih procesov nista eksplicitno opredeljeni, je mogoče pokazati s tole informacijsko formulo:

informacijska_rekurenca
pomeni_da
informacija nastaja_iz informacije
kot protiinformacija potem_se
protiinformacija vmešča_v informacijo
in_pri_tem je
nastajanje_in_vmeščevanje
informacije spontano;

V tej formuli je operator "potem_se" očitno serijski glede na predhodni del formule. Operator "nastajanje_in_vmeščanje" je paralelno strukturiran in z njim sta nastajanje in vmeščevanje informacije razumevana kot paralelna informacijska procesa. Seveda pa je mogoče paralelnost in serialnost izraziti z vpeljavo posebnih operacij (npr. "paralelno_k", "je_v_zaporedju_z" itd.). Operatorski del "je", ki sledi operatorju "in_pri_tem", spada k "spontano" na koncu formule. Tako je operator "je ... spontano" primer distribuiranega operatorja. □

Formula splošne informacijske rekurence

Prejšnji primer verbalne formule je mogoče zapisati simbolično. Naj označuje > informacijsko entiteto "informacijska_rekurenca", α informacijo, ω protiinformacijo, $\mathfrak{E}$ vmeščanje ter $\mathfrak{I}_\alpha$, $\mathfrak{E}_\alpha$ "nastajanje_in_vmeščanje informacije". Operatorji $\models$, $\models_{\mathfrak{E}}$ in $\models_{spon}$ imajo pomene "nastaja_iz", "kot", "vmešča_v" in "je_spontano". Tu so še logični operatorji $=$, $\wedge$ in $\Rightarrow$ s standardnimi pomeni "pomeni_da", "in_pri_tem" in "potem_se". Eden od možnih simbolnih zapisov je potem tale:

$$(\rho = (((\alpha \models \alpha) \models \omega) \Rightarrow (\omega \models_{\mathfrak{E}} \alpha))) \wedge (\mathfrak{I}_\alpha, \mathfrak{E}_\alpha \models_{spon})$$

Formalisti seveda lahko trdijo, da je ta zapis ne le preglednejši od njegovega verbalnega primerka, temveč da tudi več pove, ko eksplicitno opozarja (izraža) specifično, tj. informacijsko rekurenco. ■

Princip informacijske paralelnosti

Informacijska paralelnost je cirkularna razvejenost ali razcepljenost informacije, ki je prostorska (topološka), časovna (temporalna), alternativna (pluralna), multiserijska, paralelno generativna, paralelno spoznavna itd. Informacijska paralelnost je tudi sama infor-

macija. □

Formule informacijske paralelnosti

Za eksplicitno izražanje informacijske paralelnosti smo predvideli dva posebna tipa informacijskih operatorjev, in sicer $\Vdash$ in $\nVdash$ za splošno paralelnost in $\vdash$ in $\dashv$ za paralelno cikličnost. Seveda pa je mogoče eksplicirati z operatorji tudi specifično neparalelnost, in sicer splošno neparalelnost ($\nVdash$, $\Vdash$) in neparalelno necikličnost ($\nvdash$, $\dashv$). ■

Posledica informacijske paralelnosti

Obstajanje in nastajanje paralelne informacije potrebuje možnost paralelnega pojavljanja informacijskih oblik in informacijskih procesov. V tem primeru pomeni beseda "možnost" pojem paralelne strukture, ki omogoča paralelno informacijsko pojavnost. Strukturna paralelnost je na določen način prostorska, časovna, pluralna, spontana itd. Obstajanje in nastajanje paralelne informacijske strukture kot informacije je potreben pogoj informacijske paralelnosti. To velja dobesedno za paralelno informacijo v živem, kjer je informacijski paralelizem mogoč le v paralelni biološki strukturi. □

Populacija živih bitij je dober primer informacijskega območja, v katerem se paralelna informacija pojavlja spontano in raznovrstno. Ta paralelna informacija vpliva na bitja in njihovo interakcijo, tako da so ta bitja do določene stopnje informacijsko prepletena. Ta primer kaže, kako je paralelna informacija kot informacijska populacijska enota nepredvidljiva v svojem obstajanju in nastajanju, odvisna od informacijskih okoliščin in od nastajanja okoliške informacije. Paralelnost informacije se pojavlja, ko se obstoječe informacijske poti (oblike in procesi) razcepljajo v raznovrstne (pluralne) nastajalne smeri.

Primer dveh paralelnih procesov

Uvedimo dve različni informacijski entiteti, ki ju označimo z ip_1 in ip_2, označujeta pa naj dva paralelna informacijska procesa. Ta procesa naj informirata medsebojno neodvisno, vendar naj se nastala protiinformacija teh procesov vmešča v prvi in v drugi proces. Informiranje procesov ip_1 in ip_2 naj bo odvisno le od njune lastne protiinformacije. Konstruiramo lahko tole formulo:

ip_1 je_paralelno_k ip_2
na_način_ko
protiinformacija (od) ip_1
vpliva_na
informacijo (od) ip_1 in ip_2 in_na
informiranje (od) ip_1
in_paralelno_k_temu
protiinformacija (od) ip_2
vpliva_na
informacijo (od) ip_1 in ip_2 in_na
informiranje (od) ip_2;

Do tu je ta formula, ki opisuje predhodni tekst, jasno razumljiva. Operator "vpliva_na" ima lahko natančnejši pomen, in sicer "se_vmešča_v". □

Formula dveh paralelnih procesov

Označimo prvi in drugi informacijski proces z φ in ψ. Pri tem naj pomeni $\|$ "je_paralelno_k", $=$ "na_način_ko", $\omega(\varphi)$ "protiinformacija (od) φ", $\Vdash$ "vpliva_na", $\alpha(\varphi)$ "informacija (od) φ", $\alpha(\psi)$ "informacija (od) ψ", $\Im(\varphi)$ "informiranje (od) φ", $\|$ "in_paralelno_k_temu", $\omega(\psi)$ "protiinformacija (od) ψ" in $\Im(\psi)$ "informiranje (od) ψ. Simbolni zapis prejšnje verbalne formule je tedaj lahko tale:

$$(\varphi \,\|\, \psi) = ((\omega(\varphi) \Vdash \alpha(\varphi), \alpha(\psi), \Im(\varphi)) \,\|\, (\omega(\psi) \Vdash \alpha(\varphi), \alpha(\psi), \Im(\psi)))$$

To formulo lahko zapišemo poenostavljeno tako, da vzamemo informacijo α procesa φ, t.j. $\alpha(\varphi)$ kar kot proces φ. Razen tega nam splošni paralelni operator $\Vdash$ že sam po sebi zagotavlja paralelno prepletenost z drugimi procesi v paralelnem sistemu. Okrajšana oblika zadnje formule bi tako lahko bila tudi

$$(\varphi \,\|\, \psi) = ((\omega(\varphi) \Vdash \varphi, \psi, \Im(\varphi)); (\omega(\psi) \Vdash \varphi, \psi, \Im(\psi)))$$

To pa je klasična predstavitev paralelnega informacijskega sistema. Najkrajša možna simbolna predstavitev paralelnega sistema dveh procesov bi bila preprosto $\varphi \Vdash \psi$. Ta paralelni proces pa bi zopet lahko razstavili (paralelno dekomponirali) v podrobnejše procese. ■

Princip informacijske zaporednosti (serialnosti)

Informacijska zaporednost (posledičnost) je časovna, zančna, enosmerna, spominska, prostorska, zaprta itd., spontano nastajajoča cirkularnost informacije. Informacijska zaporednost je tudi sama informacija. □

Posledica informacijske zaporednosti

Obstajanje in nastajanje serijske informacije potrebuje možnost serijskega pojavljanja, interpolacije, vstavljanja (vrinjanja) informacije med informacijo in serijske rekonfiguracije serijske informacije. Čeprav je informacijska zaporednost prostorsko in časovno strukturalna, jo je mogoče razumevati prenešeno tudi na paralelen način. Zaporednost informacije kaže, da je mogoče časovnost in čas razumevati kot informacijsko razliko, ki se pojavlja v relaciji med informacijo in protiinformacijo. Ni jasne razločitve med informacijsko zaporednostjo in informacijsko paralelnostjo (vštricnostjo), vendar je lahko v nekaterih primerih razločevanje smiselno. □

Primer serijskega in paralelnega vpraševanja

Predpostavimo tri serijske procese, ki so označeni kot vprašanje, vpraševanje in vprašano (odgovorjeno). Proces vprašanja producira vprašanja kot protiinformacijo in ta protiinforma-

cija se vmešča v informacijo procesa vprašanja in procesa vpraševanja in v njuni informiranji. Proces vpraševanja producira vpraševanje kot svojo protiinformacijo in ta protiinformacija se vmešča v informacijo procesa vpraševanja in vprašanega in v njuni informiranji. Proces vprašanega producira vprašano kot svojo protiinformacijo in ta protiinformacija se vmešča v informacijo procesov vprašanega, vprašanja in vpraševanja in v njihova informiranja. Ta procesna shema je cirkularna glede na vprašanje in vpraševanje, glede na vpraševanje in vprašano ter glede na vprašano in vprašanje. Postavimo lahko tole formulo:

serijski_proces_vprašanja_vpraševanja_
_vprašanega
je

vprašanje informira in
protiinformacija_vprašanja
se_vmešča_v
informacijo (od) vprašanja, vpraševanja
in_v
informiranje (od) vprašanja, vpraševanja

potem_pa
vpraševanje informira in
protiinformacija_vpraševanja
se_vmešča_v
informacijo (od) vpraševanja, vprašanega
in_v
informiranje (od) vpraševanja, vprašanega

potem_pa
vprašano informira in
protiinformacija_vprašanega
se_vmešča_v
informacijo (od) vprašanega, vprašanja
in_v
informiranje (od) vprašanega, vprašanja

potem_pa
serijski_proces_vprašanja_vpraševanja_
_vprašanega
se_ponovi;

Paralelna informacijska formula, kjer so vprašanja, vpraševanje in vprašano paralelni procesi, je tale:

paralelni_proces_vprašanja_vpraševanja_
_vprašanega
je
vprašanje, vpraševanje, vprašano
informirajo in
protiinformacija (od) vprašanja, vpraševanja,
vprašanega
se_vmešča_v
informacijo in informiranje (od)
vprašanja, vpraševanja, vprašanega;

Cirkularnost te formule je izražena s postfiksnim operatorjem (nedovršnikom) g"informirajo". □

Formule serijskega in paralelnega vpraševanja

Prejšnja primera vpraševanja lahko seveda opišemo tudi formalno. Naj bo $\sigma_{\rightarrow}$ serijski_proces_vprašanja_vpraševanja_vprašanega, = "je", ξ vprašanje, $\wedge$ "in", $\llcorner$ povzroči nastanek, ω_{ξ} protiinformacija_vprašanja, $\models_{\mathfrak{E}}$ se_vmešča_v, $\alpha(\xi)$ informacija (od) vprašanja, η vpraševanje, $\alpha(\eta)$ informacija (od) vpraševanja, "," in_v, $\mathfrak{I}_{\xi}$ informiranje (od) vprašanja, $\mathfrak{I}_{\eta}$ informiranje (od) vpraševanja, $\rightarrow$ potem_pa, ω_{η} protiinformacija_vpraševanja, ζ vprašano, $\alpha(\zeta)$ informacija (od) vprašanega, $\mathfrak{I}_{\zeta}$ informiranje (od) vprašanega in ω_{ζ} protiinformacija_vprašanega. Serijski proces vprašanja, vpraševanja in vprašanega je potem tale:

$$(\sigma_{\rightarrow} = ((((\xi \models) \wedge ((\xi \llcorner \omega_{\xi}) \models_{\mathfrak{E}} \alpha(\xi), \alpha(\eta), \mathfrak{I}_{\xi}, \mathfrak{I}_{\eta})) \rightarrow ((\eta \models) \wedge ((\eta \llcorner \omega_{\eta}) \models_{\mathfrak{E}} \alpha(\eta), \alpha(\zeta), \mathfrak{I}_{\eta}, \mathfrak{I}_{\zeta}))) \rightarrow ((\zeta \models) \wedge ((\zeta \llcorner \omega_{\zeta}) \models_{\mathfrak{E}} \alpha(\zeta), \alpha(\xi), \mathfrak{I}_{\zeta}, \mathfrak{I}_{\xi}))) \models \sigma_{\rightarrow}$$

Označimo proces desno od operatorja = s đ. Potem imamo za gornji proces vpraševanja:

$$(\sigma_{\rightarrow} = \mathfrak{P}) \models \sigma_{\rightarrow}$$

Tu pravzaprav vidimo, kako lahko neka dinamična definicija $\sigma_{\rightarrow}$ sama informira svojo opredeljivost. Za serijsko vpraševanje bi bil mogoč tudi t.i. sistemski zapis, in sicer v obliki

$$(\sigma_{\rightarrow} = (\xi \models; (\xi \llcorner \omega_{\xi}) \models_{\mathfrak{E}} \alpha(\xi), \alpha(\eta), \mathfrak{I}_{\xi}, \mathfrak{I}_{\eta}; \eta \models; (\eta \llcorner \omega_{\eta}) \models_{\mathfrak{E}} \alpha(\eta), \alpha(\zeta), \mathfrak{I}_{\eta}, \mathfrak{I}_{\zeta}; \zeta \models; (\zeta \llcorner \omega_{\zeta}) \models_{\mathfrak{E}} \alpha(\zeta), \alpha(\xi), \mathfrak{I}_{\zeta}, \mathfrak{I}_{\xi})) \models \sigma_{\rightarrow}$$

Za paralelni proces (sistem) vpraševanja $\sigma_{\|}$ pa imamo:

$$(\sigma_{\|} = (\xi, \eta, \zeta \models; (\xi, \eta, \zeta \llcorner_{\|} \omega_{\xi}, \omega_{\eta}, \omega_{\zeta}) \models_{\mathfrak{E}} \xi, \eta, \zeta, \mathfrak{I}_{\xi}, \mathfrak{I}_{\eta}, \mathfrak{I}_{\zeta})) \models \sigma_{\|} \blacksquare$$

INFORMACIJSKE OBLIKE IN INFORMACIJSKI PROCESI

Če se informacija razumeva kot biti_v_obliki in biti_v_procesu, je smiselna uvedba pojmov informacijske oblike in informacijskega procesa. Biti_v_obliki zadeva katero koli predstavljivo obliko kot informacijo. Biti_v_procesu zadeva katerikoli predstavljivi pojav ali pojavno kompleksnost kot informacijo. Ta način gledanja je sprejemljiv za živa bitja, s katerim se tudi živo bitje samo lahko razumeva kot skupek informacijskih oblik in informacijskih procesov, ki informirajo sami sebe in so informirani s tistimi zunanjimi informacijskimi oblikami in informacijskimi procesi, ki jih zadevajo (npr. biološko vznemirjajo). S to predpostavko je mogoče sprejeti naslednje principe.

Princip informacijske oblike in informacijskega procesa

Vesoljno živo in neživo pojavnost je mogoče

predpostavljati, zaznavati, opazovati, raziskovati, spoznavati, razumevati itd. kot pojavnost (fenomenologijo) informacijskih oblik in informacijskih procesov. Oblike in procesi fizičnega in psihičnega sveta informirajo in so medsebojno informirani, tj. vznemirjajo pasivno in aktivno drug drugega in sami sebe. Vsaka pojavnost je pojavnost samo do te mere, s katero se pojavne oblike in pojavni procesi zaznavajo kot informacija in informiranje informacije. □

Ta princip nudi posebno, informacijsko usmeritev in je lahko začetek neke nove tradicije, s katero se vse razumeva kot informacijsko. To je princip informacionizma ali informizma, kot ga je opredelil avtor [IDI]. Če se karkoli pojmuje kot informacija, da je informacijska oblika ali informacijski proces, potem se to območje razumevanja imenuje informacionizem. Informizem pa ni kaj drugega, kot je mišljenje in razumevanje vesoljne pojavnosti skozi način, kako informacija informira živo in neživo.

Posledica informacijske oblike

Informacijska oblika konstituira informacijo, ki jo daje s svojim informiranjem ali sprejema iz informacijskega vira. Čeprav je informacijska oblika statičen pojav, lahko povzroča informiranje v sprejemniku. Tako informacijska oblika informira in ostra distinkcija med informacijsko obliko in informacijskim procesom ni mogoča. Informiranje informacijske oblike je možnost njenega opazovanja, raziskovanja in spoznavanja, ko informacijska oblika proizvaja njej pripadajočo protiinformacijo. V tem pomenu je informacijska oblika zelo splošen informacijski princip in je tudi sama informacija. □

Formula, ki razkriva informacijsko obliko

Informacija α razkriva oziroma lahko informira (možna implikacija $\Rightarrow_\pi$) tudi svojo informacijsko obliko φ, torej vobče $\alpha \models \varphi$. Kadar α informira informacijo β, se v β lahko reflektira tudi oblika (npr. struktura) φ_α informacije α. Torej

$$(\alpha \models \beta) \Rightarrow_\pi (\varphi_{\alpha,\beta} \subset \beta)$$

S operandom $\varphi_{\alpha,\beta}$ se dejansko nakazuje proces

$$(\alpha \models \varphi) \models \beta \blacksquare$$

Posledica informacijskega procesa

Informacijski proces je dinamična informacijska upodobitev nekega pojava, je proces v okviru informacije, ki aktivno in pasivno informira informacijo v njej sami kot procesu. Informacijski proces spreminja samega sebe s svojim samoinformiranjem in z informiranjem drugih informacijskih procesov. Sam generira protiinformacijo in sprejema to protiinformacijo v svoje lastno informacijsko upoštevanje. Informacijski proces generira informacijske oblike in informacijske procese znotraj in zunaj sebe. Vsak informacijski proces je tudi sam informacija. □

Formula, ki razkriva informacijski proces

Informacija α razkriva oziroma lahko informira (možna implikacija $\Rightarrow_\pi$) tudi svoj informacijski proces ψ (svoje informiranje oziroma informacijsko organizacijo), torej vobče $\alpha \models \psi$. Ko α informira informacijo β, se v β lahko prenaša tudi informiranje ψ informacije α. Imamo

$$(\alpha \models \beta) \Rightarrow_\pi (\psi_{\alpha,\beta} \subset \beta)$$

Pri tem lahko upoštevamo:

$$((\alpha \models \beta) \equiv (\alpha \models_\alpha) \models_\beta \beta) \equiv \alpha \models_\alpha \circ \models_\beta \beta$$

Z operandom $\psi_{\alpha,\beta}$ se dejansko nakazuje zapisani proces. ■

STRUKTURIRANJE IN ORGANIZIRANJE INFORMACIJE

Kakšno je fundamentalno razlikovanje med strukturo in organizacijo informacije? Ali je stroga ločitev med njima sploh mogoča?

Princip informacijske strukture in informacijske organizacije

Informacijska struktura je konstitucija informacije, tj. sestavljenost informacijskih oblik in informacijskih procesov, ki so povezani kot informacija. Te oblike in ti procesi so informacijske komponente. Informacijske relacije med informacijskimi komponentami, s katerimi je opredeljena sestavljena informacija, pomenijo informacijsko organizacijo. V okviru informacijske epistemologije je informacijska struktura bližje obliki, medtem ko je informacijska organizacija bližje procesu. V okviru informacije so informacijske oblike in informacijski procesi informacijsko prepletene sestavine. Informacijske sestavine informacijsko integrirajo informacijo. Informacijska struktura in informacijska organizacija sta tudi sami informacija. □

Posledica informacijske strukture in informacijske organizacije

Informacijska struktura je informacija, ki zadeva tip in obstoj informacijskih komponent. Informacijska organizacija je informacija, ki zadeva tip in obstoj informacijskih relacij, odvisnosti, nujnosti, možnosti itd. med komponentami, ki informacijo sestavljajo. S tega gledišča je informacija informacijsko strukturiran in organiziran pojav. Informacijska struktura in informacijska organizacija informirata (tj. nastajata, vmeščata, sta vmeščevani, protiinformirata) na cirkularen način (tj. rekurentno, paralelno, serijsko).

Obstaja esencialna razlika med informacijsko strukturo in informacijsko organizacijo na eni strani in med običajno, statično pojmovano strukturo in organizacijo na drugi strani. Struktura in organizacija informacije nastajata kot informacija. Zato je informacija kot enota glede na njeno strukturo in organizacijo spremenljiva in nastajajoča (prihajajoča v obstaja-

nje) z novimi informacijskimi komponentami, relacijami itd., ki se vmeščajo v obstoječo informacijo. Obstoječa informacija, ki informira kot dano (obstoječe) informacijsko ozadje, se spreminja in nastaja. Očitno je odprtost ali zaprtost informacije kot enote in kot avtonomije posledica informacijskega strukturiranja in organiziranja. □

Formula informacijske strukture

Informacija α je vobče sestavljena iz informacijskih oblik in procesov, tj. iz svojih komponent. Ta sestavljenost σ je tudi sama informacija oziroma informacijska komponenta. Informacija α lahko tako informira tudi svojo sestavljenost ali strukturo σ, torej $\alpha \models \sigma$. Sestavljenost σ pove, kako je α konstituirano. Če je α informacijski proces, ki je predstavljiv z informacijsko formulo φ, potem v okviru formule φ opisljivi podprocesi (podformule) sestavljajo strukturo σ. Tako je npr. $\sigma(\alpha)$ lahko množica podprocesov formule $\varphi(\alpha)$, ki opisuje α.

Naj bo npr. formula φ opis procesa α. Imamo metaformulo $\alpha = \varphi$. Naj bo formula φ izraz

$$(\xi \models \eta) \Vdash \zeta$$

V tem primeru je σ množica procesov (njihovih formul), in sicer

$$\sigma = \{\xi, \eta, \zeta, \xi \models \eta, (\xi \models \eta), (\xi \models \eta) \Vdash \zeta\}$$

Z dekompozicijo procesa α oziroma njegovih podprocesov, tj. formule φ oziroma njenih podformul, je mogoče dobiti bolj in bolj podrobno strukturo σ. ■

Oglejmo si tale primer! Formula φ je program, α pa je izvajanje (računalniško izvrševanje) programa φ. Nadalje naj bo σ struktura programa φ. V njej se pojavljajo npr. različni podprogrami, tj. procedure, zanke, if-stavki, podatkovne deklaracije itd. Struktura σ je tedaj konstitualna analiza programa φ za programski proces α.

Formula informacijske organizacije

Organizacija ω informacije α so informacijske relacije med informacijskimi komponentami informacije α. Naj bo φ formula za α. Organizacija $\omega(\alpha)$ je npr. operacijska analiza formule φ za α, ki lahko zajema tako eksplicitne kot implicitne informacijske operatorje. Naj bo φ zopet $(\xi \models \eta) \Vdash \zeta$. Lahko imamo primer:

$$\omega = \{\models, \Vdash, \Im_{\xi}, \Im_{\eta}, \complement_{(\xi \models \eta)}\}$$

Entiteti $\Im$ in $\complement$ označujeta implicitno informiranje in protiinformiranje. Z nadaljno dekompozicijo formule φ postaja tudi ω bolj raznovrstno. Pri ω lahko govorimo podobno kot pri σ o partikularizaciji in univerzalizaciji ω in σ, pa tudi o možnosti njune analitične razširljivosti. ■

Primeri kulturnih oblik

Kulturne oblike (npr. filozofija, etika, ideologija, znanost, umetnost itd.) kot informacija so evidentni primeri partikularnih informacijskih struktur in informacijskih organizacij. Strukturno in organizacijsko partikularnost kulturne oblike imenujemo informacijsko postavje (enframing, Gestell). Tako postavje, ki je regularno vmeščeno v informacijo, obstaja tudi izven partikularne kulturne oblike. Na splošno je informacijsko postavje neke kulturne oblike vmeščeno v širše področje informacije, ki oblikuje ozadje danega postavja. Npr. matematika je kot disciplina postavljena v matematično strukturo in organizacijo, tj. v svoje matematično postavje. To postavje je vmeščeno v širše informacijsko okolje, v več drugih znanstvenih disciplin, v filozofijo in jezik itd., v katerih matematika korenini, se uporablja, vpliva itn. □

Struktura in organizacija informacije konstituirata informacijsko postavje, ki je informacijsko vmeščeno v širši informacijski kontekst. V tem smislu ne more biti nobena znanstvena disciplina ali kulturna oblika zgolj znanstvena ali zgolj kulturna, tj. zaprta zgolj sama vase kot znanstvena disciplina ali kulturna oblika. Informacijsko postavje je princip informacijskega strukturiranja in organiziranja in je tudi samo informacija.

INTELIGENCA KOT INFORMACIJA

> ... Bistvo inteligence je, da deluje primerno, ko ni enostavne vnaprejšnje opredelitve problema ali prostora stanj, v katerem bi se iskala rešitev. ... Heidegger pokaže, da je bistvo naše inteligence v naši vrženosti in ne v naši refleksiji. Podobno pokaže Maturana, da biološki spoznavni sistemi ne delujejo z manipulacijo predstavitev zunanjega sveta.
>
> Winograd in Flores [UCC] 98, 99

V splošnem ni inteligenca kot informacija ničesar drugega kot neka informacijska kompleksnost, namenska, intencionalna in ciljno usmerjena sestavljenost informacije. Inteligenca je tako informacijski produkt, narejen z divergentnimi, višjimi informacijskimi funkcijami živega informacijskega sistema. Informacija konstruira inteligenco z namenom, da bi bila ta inteligenca uporabljiva kot primerna informacija v distinktnih položajih bivanja bitja, pri reševanju njegovih realnih in umetnih problemov.

Princip inteligence

Inteligenca, v kateri je vmeščena informacija intencionalna, informacijsko postavljiva (uokvirjena), utemeljena informacija (tj. s konsekventno strukturo in organizacijo), pripadajoča specifični domeni (disciplini, področju), se razvija (nastaja) na informacijsko površje za specifično reševanje problemov (vedenje, mišljenje, delovanje). Inteligenca pojmuje (v opazovanju, raziskovanju, razpoznavanju) le specifično problemsko domeno in le v okviru te domene vpliva inteligentno na vedenje in akci-

jo. Inteligenca je specializirana informacija, ki je informacijsko postavljena (uokvirjena), zadeva zaprto (tesno) neko problemsko področje in spoznava striktno le v metadomeni neke problemske domene. Zaradi tega potrebuje inteligenca drugo informacijo, celovito spoznavanje (procesiranje) druge informacije, šolanje (vadbo), znanje in naposled značilno informacijsko ekspertizo. □

Posledica inteligence kot informacije

Inteligenca je informacija, ki je zožena v specifično problemsko domeno, s svojo lastno reševalno zmogljivostjo, izkustvom in značilno metodološko slepoto. Inteligenca pozna svoj lasten sistematičen pristop, svojo pot rekurentnega sestopa. V spoznavni metadomeni inteligence se nahajajo specifična pravila, znanje, ekspertiza, metodologija, konceptualna spretnost in njej lastna specializirana orodja in procedure reševanja problemov. Inteligenca vselej potrebuje svojo značilno informacijsko substanco, iz katere nastaja in v katero se lahko vmešča. Nastajanje inteligence je odvisno od inteligentnega okolja, obdajajočega sveta, v katerem je inteligenca vdomljena in iz katerega lahko raste. □

Formule, ki izražajo inteligenco

Kaj je mogoče na formalen način povedati o inteligenci kot informacijskem pojavu? Ali je mogoče že iz same strukture formule zaslediti inteligentnost procesa, ki ga formula opisuje? Ali obstaja kakšen poseben način sklepanja (modus informationis), ki je v svojem bistvu že inteligenčen način informiranja?

Inteligenca η nastaja iz svojega informacijskega okolja α in v tem kontekstu velja gotovo $\alpha \models \eta$. Vendar inteligenca η hkrati vpliva povratno tudi na svoje izhodišče, iz katerega nastaja in s tem lahko ojačuje svojo kompleksnost, moč, spremenljivost, prilagodljivost, skratka lastno naravo in stopnjo svoje inteligentnosti. Za inteligenco je tedaj značilna "dinamična" oblika formule, in sicer

$$\ldots((\ldots\ ((\alpha \models \eta) \models \alpha) \models \eta\ \ldots) \models \alpha) \models \eta\ \ldots$$

Inteligenca η ima tudi svojo specifiko, specialnost oziroma intencionalnost $\Re$ (namernost, informacijsko usmerjeno razvojnost k določeni specifičnosti, racionalnost itd.), ki je njeno notranje gonilo, implicitno jedro, tako da lahko zapišemo formalno $\Re(\eta) \subset \eta$. To jedro se razvija skupaj z inteligenco η in okoliško informacijo α, pri čemer se η in $\Re(\eta)$ medsebojno informacijsko prepletata. Intenca $\Re(\eta)$ lahko deluje kot cilj inteligence η, kot njena vodilna informacija. Tako pomeni $\Re(\eta)$, da obstaja informacijski proces, ki je opisljiv s formulo

$$\ldots\ ((\eta \models \Re(\eta)) \models \eta) \models \Re(\eta)\ \ldots$$

paralelno k procesu, opisanem s prejšnjo formulo o nastajanju inteligence η iz informacije α itd. Zapis splošnega paralelnega sistema inteligence bi bil

$$\alpha,\ \eta,\ \Re(\eta) \models \alpha,\ \eta,\ \Re(\eta)$$

kjer vsaka operandna komponenta informacijsko vpliva na vsako operandno komponento. Ta formula je tedaj ne le okrajšani zapis za devet paralelnih formul ($\alpha \models \alpha$; $\alpha \models \eta$; $\alpha \models \Re(\eta)$; $\eta \models \alpha$; $\eta \models \eta$; $\eta \models \Re(\eta)$; $\Re(\eta) \models \alpha$; $\Re(\eta) \models \eta$; $\Re(\eta) \models \Re(\eta)$), temveč je iz nje mogoče izpeljati tudi prejšnji "inteligenčni" formuli. Uporaba transformacijskega pravila iz informacijske domene modus rectus [IL4] pa omogoča, da v danem informacijskem procesu, opisanim s formulo, iščemo tudi inteligenco oziroma njeno implicitno jedro kot implicitno informacijo v informacijskem procesu. ■

INFORMACIJSKI STROJ IN INFORMACIJSKI PROGRAM

> ... Četudi bi računalniški program vselej zmagoval v šahovski igri, bi Winograd in Flores trdila, da to ni inteligenca.
>
> Clancey [AI] 243-244

Pojem informacijskega stroja in informacijskega programa vznemirja današnjo filozofijo, znanost in tehnologijo. Vprašanje razvoja stroja in programa prihodnosti naj bi bilo dosledno utemeljeno v informacijskem načinu gledanja, tj. v načinu filozofskih in realizacijskih možnost strojev in programov kot informacijskih entitet.

Princip informacijskega stroja

Stroj je informacijski, če sta njegova struktura (substanca, oblika, strukturne komponente) in njegova organizacija (strojna povezanost, relacije, procesiranje) informacijski. Struktura stroja zadeva arhitekturo stroja (fizično, biološko, kostitucijsko). Organizacija stroja zadeva delovanje stroja (um, vedenje) in funkcionalno fleksibilnost (možnosti krmiljenja, programiranja) v raznovrstnih notranjih in zunanjih okoliščinah. Arhitektura informacijskega stroja je dinamična (možganska), je dinamično krmiljena, spremenljiva, nastajajoča in odvisna od obdajajočega sveta. Informacijski stroj je lahko upravljan z informacijskim programom, da bi tako dosegal informacijske značilnosti in lastnosti. Informacijski stroj deluje kot informacija. □

Posledica principa informacijskega stroja

Ker informacijski stroj poseduje informacijske atribute (informiranje, vmeščevanje, nastajanje, protiinformiranje v informacijsko cirkularni in spontani obliki), je njegovo delovanje informacijsko. To delovanje je dosegljivo z njegovo informacijsko arhitekturo in z njegovo informacijsko programiranostjo. Dinamična arhitektura (arhitekturno preklapljanje brez arhitekturnega nastajanja) pomeni seveda bistveno omejenost informacijskega stroja. To slabost njegove informacijske dinamike je mogoče delno kompenzirati z informacijskimi programi. Informacijski stroj, ki bi imel polno (ali celo le delno) informacijsko moč zaenkrat tehnološko še ni uresničljiv. □

Formule informacijskega stroja

Informacijski stroj $\mathfrak{S}$ ima vobče dve sestavljeni komponenti: arhitekturo ali strukturo σ in programsko informacijo π. Arhitektura σ stroja $\mathfrak{S}$ je tista temeljna informacijska podstat, torej temeljna informacija, v katero je mogoče vmeščati drugo dopolnilno arhitekturno kot operacijsko oziroma programsko informacijo. Programska informacija π stroja $\mathfrak{S}$ ima lahko več ravnin oziroma komponent, in sicer osnovno ali bazično komponento β (npr. neke vrste informacijski ROM), operacijsko komponento ω (informacijski operacijski sistem z ustreznimi pripomočki za lasten razvoj oziroma nastajanje) in aplikativno komponento α (trenutni aplikativni informacijski segmenti). Informacijski stroj je paralelni konstrukt, ki ga simbolično označimo kot

$$\mathfrak{S} = (\sigma \parallel \pi)$$

Informacijski operator $\parallel$ je v kontekstu informacijskega stroja $\mathfrak{S}$ opredeljen takole:

$$(\sigma \parallel \pi) =_{Df} (\Diamond\sigma.((\sigma \in \mathfrak{A}) \rightarrow (\Diamond\pi.(\pi \gg \sigma))))$$

V tej formuli pomeni $\mathfrak{A}$ vse možne arhitekture informacijskega stroja, operator $\Diamond$ beremo kot "je_mogoče" inoperator $\rightarrow$ kot "pogojuje". Operator . lahko v tem primeru beremo kot "kot" (ali tak_...,_da_je) in operator $\gg$ kot "je_prilagojeno". Operator $\parallel$ tako ne pomeni le paralelnosti, temveč izraža tudi določeno pogojnost med operandi (operator $\rightarrow$). Zaradi jasnosti preberimo definicijo informacijskega stroja takole: informacijska struktura σ je paralelna k programski informaciji π tedaj in le tedaj, ko obstaja takšna struktura σ, da če je σ iz razreda možnih informacijskih aritektur $\mathfrak{A}$, potem je močna tudi programska informacija π, ki je prilagojena arhitekturi σ, tj. $\pi \gg \sigma$.

Nadalje je programska komponenta π stroja $\mathfrak{S}$ paralelni konstrukt, označen z $\pi = ((\beta \parallel \omega) \parallel \alpha)$. Ta konstrukt je lahko opredeljen takole:

$$((\beta \parallel \omega) \parallel \alpha) =_{Df}$$

$$(\exists\beta.(((\beta \gg (\pi \gg \sigma)) \rightarrow (\Diamond\omega.((\omega \gg \beta) \gg \sigma))) \rightarrow (\Diamond\alpha.(((\alpha \gg \omega) \gg \beta) \gg \sigma)))$$

Stroj $\mathfrak{S}$ je nastajalen v vseh svojih komponentah, tj. v σ in π oziroma podrobneje v β, ω in α. Komponente α, ω in α so informacijski programi, torej informacija. ■

Formalne možnosti inteligentnega stroja

Označimo z $\eta_{\mathfrak{S}}$ inteligenco informacijskega stroja $\mathfrak{S}$, ki nastaja iz svojega informacijskega okolja $\mathfrak{S} = (\sigma \parallel \pi)$ in $\pi = ((\beta \parallel \omega) \parallel \alpha)$, ko velja $\mathfrak{S} \models \eta_{\mathfrak{S}}$. Strojna inteligenca $\eta_{\mathfrak{S}}$ pa povratno vpliva tudi na stroj $\mathfrak{S}$, iz katerega nastaja. Osnovna formula inteligentnega stroja je tedaj

$$\ldots(\ldots((\mathfrak{S} \models \eta_{\mathfrak{S}}) \models \mathfrak{S}) \models \eta_{\mathfrak{S}} \ldots) \models \mathfrak{S} \ldots$$

kjer je $\mathfrak{S} = (\sigma \parallel ((\beta \parallel \omega) \parallel \alpha))$. Tu se pokaže možna kompleksnost inteligentnega stroja $\mathfrak{S}$. Strojna inteligenca ima svojo specifiko (intencionalnost, jedro) $\Re(\eta_{\mathfrak{S}}) \subset \mathfrak{S}$. Strojna intenca $\Re(\eta_{\mathfrak{S}})$ lahko deluje kot cilj strojne inteligence $\eta_{\mathfrak{S}}$, kot njena strukturirna, arhitekturo in programsko informacijo oblikovalna informacija. Tako pomeni $\Re(\eta_{\mathfrak{S}})$, da obstaja informacijski proces, ki oblikuje stroj in njegove programe in je opisljiv s formulo

$$\ldots((\eta_{\mathfrak{S}} \models \Re(\eta_{\mathfrak{S}})) \models \eta_{\mathfrak{S}}) \models \Re(\eta_{\mathfrak{S}}) \ldots$$

paralelno k procesu, opisanem s prejšnjo formulo o nastajanju inteligence $\eta_{\mathfrak{S}}$ iz informacije $\mathfrak{S}$ itd. Zapis splošnega paralelnega sistema strojne inteligence bi bil

$$\mathfrak{S},\ \eta_{\mathfrak{S}},\ \Re(\eta_{\mathfrak{S}}) \models \mathfrak{S},\ \eta_{\mathfrak{S}},\ \Re(\eta_{\mathfrak{S}})$$

kjer vsaka operandna komponenta oziroma komponenta v operandu informacijsko vpliva na vsako operandno komponento. Ta formula je tedaj ne le okrajšani zapis za devet paralelnih formul ($\mathfrak{S} \models \mathfrak{S}$; $\mathfrak{S} \models \eta_{\mathfrak{S}}$; $\mathfrak{S} \models \Re(\eta_{\mathfrak{S}})$; $\eta_{\mathfrak{S}} \models \mathfrak{S}$; $\eta_{\mathfrak{S}} \models \eta_{\mathfrak{S}}$; $\eta_{\mathfrak{S}} \models \Re(\eta_{\mathfrak{S}})$; $\Re(\eta_{\mathfrak{S}}) \models \mathfrak{S}$; $\Re(\eta_{\mathfrak{S}}) \models \eta_{\mathfrak{S}}$; $\Re(\eta_{\mathfrak{S}}) \models \Re(\eta_{\mathfrak{S}})$), temveč je iz nje mogoče izpeljati tudi prejšnji "inteligenčni" formuli. V teh devetih formulah moramo seveda upoštevati strukturo stroja $\mathfrak{S} = (\sigma \parallel ((\beta \parallel \omega) \parallel \alpha))$. Uporaba raznovrstnih transformacijskih pravil iz informacijske domene modus rectus [IL4] pa omogoča, da v danem informacijskem procesu, opisanim s formulo inteligentnega stroja, iščemo tudi njegovo inteligenco oziroma inteligenčnono jedro kot implicitno informacijo v informacijskem procesu. ■

T.i. nevralna vezja oziroma nevralne mreže pa so tehnološki dosežek, ki je na poti do informacijskega stroja, saj naj bi omogočal t.i. arhitekturno genezo, tj. nastajanje strojne strukture v odvisnosti od informacijskih potreb. Nujno bo začeti še vrsto novih raziskovalnih in razvojnih projektov, da bi si izmislili gradnike informacijske arhitekture z lastnostjo raznovrstnejše informacijske spremenljivosti, še posebej paralelnosti in njune uporabnosti v dinamični arhitekturi. Te arhitekture bi se lahko časovno spreminjale, nastajale ali razvijale kot posledica informacijskih sprememb in nastajanja v arhitekturnih okoljih.

Princip informacijskega programa

Informacijski program bi bil enostavno informacija, ki spontano informira, vmešča, nastaja in protiinformira na informacijsko cirkularen način v okviru informacijskega stroja. Informacijski program informira samega sebe in druge informacijske programe in se uporablja in je vmeščen v informacijskem stroju za produkcijo informacije (npr. inteligence, namenskih informacijskih funkcij itd). □

Očitno obstaja bistvena razlika med računalniškim in informacijskim programom. Prvi je algoritmičen (matematičen, proceduralen, informacijsko statičen in definitorno predvidljiv), medtem ko je drugi informacijski (nematematičen, inteligenten, informacijsko dinamičen in definitorno nepredvidljiv). Računalniški program ima praviloma stabilno, nespremenljivo programsko strukturo in programsko organizacijo. Njegove definicije (deklaracije) ni mogoče spreminjati dinamično med njegovim izvrševanjem

s pomočjo paralelnega izvajanja njega samega in drugih programov nad samim seboj in nad drugimi delujočimi programi, podatki itd.

Posledica principa informacijskega programa

Informacijski program se vede kot informacija. V tem primeru je program tudi informacijski objekt, ki se lahko informacijsko spreminja med svojim procesiranjem drugih objektov. Značilen računalniški program se vselej vede kot subjekt, s katerim je mogoče spreminjati neprogramske objekte, nastaja pa lahko le z zunanjimi posegi. Zamisel o informatizaciji programa pripelje še do bistveno novih zahtev o programirnih orodjih, ki jih je nemogoče primerjati z današnjimi orodji za programiranje. □

Primer tehnološkega informacijskega stroja

Tehnološki (umetni) informacijski stroj naj bi imel v glavnem arhitekturo, ki je bistveno različna od arhitekture in njenih komponent današnjega računalnika. T.i. možganska ali nevralna arhitektura je le ena izmed možnosti nove usmeritve. Kaj naj bi pomenila nevralna arhitektura? V grobem so komponente nevralne arhitekture nevroni in funkcionalne enote nevronov (regije, jedra, območja, korteksi) oziroma nevronske populacije. Nevron je živčna celica (specializirani procesor) s svojimi raznovrstnimi in kompleksnimi informacijskimi procesi. V živih možganih tudi dva nevrona (ali bazična celična procesorja) nista enaka. Signali (informacija) vstopajo v nevron iz drugih nevronov prek sinaps (sinapsnih procesorjev). Nevroni lahko nastajajo pod vplivom učenja, zamenjujejo obstoječe (postarane) nevrone [BSN] in izginjajo tako funcionalno kot fizično. Sinaptične povezave med nevroni se lahko pojavljajo v odvisnosti od različnih notranjih in zunanjih strukturnih in procesnih pojavov. Nevralna arhitektura se razvija skladno z življenjskimi možnostmi pod pritiskom in z vplivom notranjih in zunanjih okoliščin. Ta arhitektura nastaja dinamično in se vede kot informacija na strukturalni (substancijski, tehnološki) in organizacijski (procesni) ravni. □

Primer informacijskega programa

Primer informacijskega programa je korteksna (nevralna) funkcija. Ta program ne deluje le kot informacija, temveč vpliva tudi na zadevno živo substanco (arhitekturo), v kateri se razvija (v kateri procesira). Tako je mogoče razumeti, da um (kot informacijski program) vpliva na razvoj možganov (kot informacijsko arhitekturo z njenimi komponentami) in da takšno nastajanje arhitekture nudi nove možnosti za nastajanje uma. □

ŽIVI PRIMERI INFORMACIJE

V prejšnjih dveh primerih smo nakazali informacijske možnosti živega informacijskega stroja (možganov) in živega informacijskega programa (uma). Maturana in Varela [AAC] sta pokazala s teoretsko obravnavo, kako je žive organizme mogoče razumevati kot avtopoezijo (autopoiesis).

Princip živega informacijskega stroja

Živ informacijski stroj predstavlja le podskupino možnih (zamisljivih) informacijskih strojev. Živ informacijski stroj je omogočen v okviru biološke strukture in organizacije, z njuno avtopoezijo. Takšen stroj je organiziran kot informacijska mreža informacijskih oblik in procesov za produkcijo, transformacijo in destrukcijo svoje lastne informacije. Živ informacijski stroj upravlja z življenjem stroja, ohranja svoje lastno življenje in producira sebe in svojo informacijo. □

Očitno imajo živi informacijski stroji kot produkti svojega lastnega delovanja nekaj, kar zadeva njih same, njihovo lastno identiteto, enotnost (lastnost informacijskega kot enote) in življensko procesiranje. Delovanje, ki se ne tiče samega stroja, je npr. značilno za umetne, alopoetične stroje. Ti stroji se sami po sebi ne razvijajo in ne ohranjajo sami sebe, toda producirajo funkcije za druge, nesvoje namene (npr. računalniki, mehanični stroji).

Princip živega informacijskega procesiranja

Živo informacijsko procesiranje je biološka pojavnost avtopoetičnih informacijskih sistemov v živem in obdajajočem svetu, in sicer do te mere, ko je to procesiranje tako ali drugače odvisno od avtopoezije (biološke samoprodukcije) ene ali več avtopoetičnih informacijskih enot. Domena vseh interakcij, v katere avtopoetičen sistem lahko vstopa, je njegova kognitivna domena. V okviru te informacijske domene ne more biti presežena avtopoetična identiteta informacijske enote in njene interakcije. □

Primeri živih informacijskih strojev in živega informacijskega procesiranja

Očitni primeri živih informacijskih strojev so molekule življenja, celice, celične populacije (organizmi), korteksna jedra, korteksi, možgani in živo bitje kot celota (enota). Živi informacijski procesi v teh strojih so npr. beljakovinska sinteza, imunost, evolucijsko učenje, evolucijske spremembe, selekcija, replikacija (razmnoževanje), um itd. □

FORMALIZACIJA INFORMACIJSKIH PRINCIPOV

V nekaterih primerih tega spisa smo že pokazali možnosti oblikovanja informacijskih formul za opisovanje informacijskih oblik in informacijskih procesov. Pokazali smo, kako je mogoče te oblike in procese formalizirati z linvističnimi pripomočki. Informacijo smo osnovali kot informacijski koncept, ki ga je mogoče zadovoljivo formalizirati le z informacijskimi sredstvi. Tudi sama informacija se kaže kot neke vrste sistem v sami sebi. V okviru tega sistema lahko informacija nastaja in se vmešča cirkularno in ima pri tem svojo strukturo,

organizacijo, oblike in procese. Informacija je vržena (metana) v informacijsko okolje iz nekega informacijskega vira, iz katerega je nastala in se razvija še naprej, nastaja, raste, propada, izginja itd. kot specifičen sistem, ki je v relaciji s samim seboj in s svojim informacijskim okoljem. S pojavitvijo informacije začne nastajati tudi njen sistem.

Vprašanje, ki se postavlja, je, kako bi bilo mogoče formalizirati sisteme, ki so informacijsko relacijski, informacijsko značilni ali informacijsko substancialni. Bolj ali manj je evidentno, da takšna formalizacija ne more več obdržati zgolj tradicionalne oblike matematične, algoritmične ali logične formalne interpretacije. Tradicionalna interpretacija je informacija, ki ne razpolaga s sredstvi, formalnim aparatom ali matematičnim formalizmom, ki bi bil pripraven (instrumentaliziran) za postavje nastajajoče informacijske pojavnosti. To pa seveda ne pomeni, da ni mogoče obstoječega matematičnega formalizma uporabljati za interpretacijo sistemov, ki so značilno terminirani, informacijsko statični in na današnji stopnji razvoja tudi še tehnološko sprejemljivi.

V predhodnih primerih nismo uporabili formalnega simbolizma, ki bi lahko zamenjeval verbalne opise v naravnem jeziku. Namen te verbalne formalizacije je bil v interpretaciji informacijskih principov z naravnimi zaporedji informacijskih operatorjev in operandov. Tako formalizirani primeri so le nakazovali pot, kako bi bilo mogoče vpeljati sistem informacijske logike, ki bi omogočal aksiomatizacijo principov in gradnjo partikularnih informacijskih teorij, ki bi lahko upoštevale pojme, kot so spontanost, cirkularnost, informacijsko nastajanje in vmeščanje, protiinformiranje, paralelnost in posledičnost informacije, informacijsko prepletenost, strukturo, organizacijo itd. Določene študije prav v tej smeri pa so bile medtem že opravljene [IL1, IL2, IL3, IL4].

Formule informacijske logike

Informacijske principe je mogoče postaviti v simbolno logično postavje. V prejšnjih primerih smo vselej lahko opazovali tipe formul, ki so bile zaporedja informacijskih operandov in informacijskih operatorjev (in ničesar drugega). Imeli smo sistem petih tipov formul, ki so bile rekurzivno razširljive do poljubne kompleksnosti. Ta logični sistem formul je bil

$$\alpha,\ \models\alpha\models,\ \alpha\models,\ \models\alpha,\ \alpha\models\alpha$$

K temu sistemu je potrebno še kratko pojasnilo. Formula α je lahko enostavna ali rekurzivno sestavljena iz formul sistema. Začnemo lahko s formulami α, ki so osnovne oblike $\models\alpha\models$, $\alpha\models$, $\models\alpha$ in $\alpha\models\alpha$. Seveda je formula tudi logični sistem formul, ki ga kot formulo moramo še pojasniti. Druga formula sistema $\models\alpha\models$ omogoča izražavo formule (α) in tudi $\alpha(\alpha)$, kjer je operator $\models$ partikulariziran kot $\models($ in kot $\models)$ oziroma kot znak "(" in znak ")". Tudi vejica in podpi~je sta npr. partikularizaciji v kombinaciji $\alpha\models$. S tem smo v glavnem pojasnili strukturo logičnega sistema formul. K temu dodajmo le še princip partikularizacije in univerzalizacije operatorjev. S tem je povedano domala vse, kar je potrebno za t.i. aksiomatizacijo sistema informacijske logike. ■

EPILOG

O metaoperatorju $\models$ in njegovi implementaciji

Uvedba pojma informacijskega metaoperatorja $\models$ in njegovih izvedenk ($\not\models$, $\models\!\!\!/$, $\not\models\!\!\!/$, $\models\!\!\!\models$, $\not\models\!\!\!\models$, $\models\!\!\!\models\!\!\!/$, $\not\models\!\!\!\models\!\!\!/$, $\vdash$, $\dashv$, $\vdash\!\!\!/$, $\dashv\!\!\!/$, $\vdash\!\!\!\vdash$, $\dashv\!\!\!\vdash$, $\vdash\!\!\!\vdash\!\!\!/$, $\dashv\!\!\!\vdash\!\!\!/$ itd.) je omogočila razvoj formalizacije koncepta informacijskega nastajanja. Ta operator se je v formulah lahko pojavljal le vezan na informacijski operand α, in sicer v sploih oblikah $\alpha\models$, $\models\alpha$ in $\alpha\models\alpha$ ali v operatorskih kompozicijah, kjer je kompozicija prevzela vlogo operatorja. Kaj pomenijo posamezna oblike izražave?

Oblika $\alpha\models$ izraža lastnost informacije α, da informira, da iz sebe oddaja ali odpošilja informacijo kot informiranje same sebe. Če bi zapisali $\alpha\models_\pi$, bi rekli, da informacija α lahko informira; vendar α vselej informira. V okviru t.i. modalne logike [ML] bi imeli tole informacijski definicijo za naš primer:

$$(\alpha\models) =_{Df} (\forall(\mathfrak{M},\ \beta\in\mathfrak{M}).(\alpha\models_{\mathfrak{M},\beta}))$$

Tu je $\mathfrak{M}$ t.i. model možnih svetov in β možni svet. Operator $\models_{\mathfrak{M},\beta}$ je tedaj že partikularna oblika operatorja $\models$. Lahko bi rekli enostavno, da je operator $\models$ določen za informacijsko domeno $(\mathfrak{M},\ \beta)$, ki je vselej lahko splošna, skladno s trenutno potrebo ali uporabo. V tem primeru ima vsaka informacija α možnost, da odpošilja informacijo, da informira. To je aktivna vloga informacije in tudi podatka. $\alpha\models$ pomeni, da α informira v vseh modelih možnih svetov in v vsakem možnem svetu modela.

Oblika $\models\alpha$ izraža lastnost informacije α, da je informirana, da je na določen način občutljiva za sprejemanje informacije kot informiranja v sami sebi. Če bi zapisali $\models_\pi\alpha$, bi rekli, da je informacija α lahko informirana. Podatek (kot posebna, omejena informacijska entiteta) po definiciji ne more biti informiran (velja npr. $\not\models\alpha$), ne more sprejemati informacije. V okviru modalne logike bi lahko postavili tole informacijsko definicijo:

$$(\models\alpha) =_{Df} (\exists(\mathfrak{M},\ \beta\in\mathfrak{M}).(\models_{\mathfrak{M},\beta}\alpha))$$

V tem primeru obstaja informacija α, ki ima možnost, da sprejema informacijo, da je informirana (v sebi informacijsko spreminjana). Ta vloga informacije je aktivnost informacijskega sprejemanja. $\models\alpha$ pomeni, da obstajajo takšni modeli možnih svetov in v njih možni svet β, da je α informirano.

Oblika $\alpha\models\alpha$ izraža lastnost informacije α, da informira in da je informirana. Definicija tega pojava bi bila

$$(\alpha\models\alpha) =_{Df} ((\forall(\mathfrak{M},\ \beta\in\mathfrak{M}).(\alpha\models_{\mathfrak{M},\beta}))\ \vee\ (\exists(\mathfrak{M},\ \beta\in\mathfrak{M}).(\models_{\mathfrak{M},\beta}\alpha))$$

Vprašanje, na katerega moramo zadovoljivo odgovoriti, je še, kako si je mogoče implementacijsko zamisliti naravo informacijskega operatorja $\models$ oziroma neko njegovo partikularizacijo. Pri enostavnih, informacijsko statičnih (matematičnih, tradicionalno logičnih) operatorjih je odgovor npr. ta, da je vselej mogoče kon-

struirati programe/operatorje, ki opravljajo funkcijo zahtevane operacije. V teh primerih še ne gre za operatorje, ki bi imeli lastnost informacijske nastajalnosti kot bistvo svoje operacijske narave.

Informacijski operator $\models$ - tudi ko je partikulariziran - mora opravljati svojo operacijo ekspertno, ko se lahko po svoji naravi vede kot "avtonomen" ekspertni sistem. "Lahko" zaradi tega, ker dinamičen ekspertni sistem ni edina operativna oblika, ki bi si jo bilo mogoče zamisliti. Operator $\models$ mora imeti dovolj generativne svobode, da se v okviru informacije razvija, postavlja, oblikuje z matično (svojo) in drugo informacijo. Tudi kompleksnost operatorja $\models$ ni omejena, je pa lahko intencionalna (npr. inteligentna). Iz tega izhodišča je mogoče raziskovati in snovati operatorje tipa $\models$ konkretno, aplikativno in funkcionalno.

Odgovor na možno kritiko koncepta informacije

Obstajajo glasni in tihi kritiki predloženega koncepta informacije oziroma informacijskega. Njim in kritičnemu umu nasploh je namenjen naslednji odgovor.

(A) Ta spis dokazuje, da je mogoče verbalno (ali filozofsko) zasnovane principe informacije tudi formalizirati na način dobre (matematične, filozofske) formalizacije (simbolizma, aksiomatizma). Zavedam pa se, da je predlagani koncept informacije filozofsko nezadosten in premalo izčrpen, ker je morda preveč konstruiran, postavljen, instrumentaliziran in verbalno še prešibek.

(B) Često prevladuje prepričanje, da ni smiselno ali potrebno razglabljati o tem, kaj informacija je. To prepričanje je dokaj razširjeno med matematiki, ki jim je informacija zgolj neko področje matematičnega postavja in med t.i. informatiki, ki razumevajo informacijo že kot danost in priročnost, o kateri se več ne razpravlja. Informacija je za njih samorazumljivi atribut realnega sveta, računalniške manipulacije in morda še dogajanja v možganih, če je temu sploh verjeti. Med matematično usmerjenimi informatiki (teorija informacije) pa je verjetno najbolj sporna predložena redefinicija informacije, ki zahteva kompleksnejši pogled na informacijsko pojavnost, kot so ga bili vajeni doslej v svojih teorijah in predstavah.

(C) Če se predloženi koncept informacije približuje tudi informacijski pojavnosti v živem, potem ta koncept lahko zadeva vrsto znanstvenih postavij, ki temeljijo npr. v kognitivnih znanostih, psihologiji, psihiatriji, nevrolingvistiki, nevrofiziologiji, umetni inteligenci itd. Ta področja navajam zato, ker sem prejel številne zahteve za separat članka "Principles of Information", objavljenega v časopisu Cyberentica. Predloženi koncept informacije je seveda šele začetek nekega postavja, ki pa se lahko razvija kot logika informacije v številne nove smeri v posameznih strokah. In tu se kaže informacijsko brezno, tudi kot odprtost za doseganje do včeraj racionalno nedosegljivih pojmov in pojmovanj.

SLOVSTVO

[ML] Chellas, B.F., Modal Logic: An Introduction, 1988 (reprinted) (Cambridge University Press, Cambridge).

[AI] Clancey, W.J., Book Review, Artificial Intelligence 31 (1987) 232-250.

[SCC] Denning, P.J., The Science of Computing, American Scientist 75 (1987) 2, 130-132.

[ESD] Gould, S.J., Ever since Darwin, 1977 (Norton, New York).

[BSN] Nottebohm, F., From Bird Song to Neurogenesis, Scientific American (1989) 2, 56-61.

[CCH] Jensen, R.V., Classical Chaos, American Scientist 75 (1987) 2, 168-181.

[AAC] Maturana, H.R., and Varela, F.J., Autopoiesis and Cognition, The Relation of the Living, 1980 (D. Reidel PC, Dodrecht, Holland).

[UCC] Winograd, T., and Flores, F., Understanding Computers and Cognition, A New Foundation for Design, 1986 (Ablex PC, Norwood, New Jersey).

[OUC] Winograd, T., On Understanding Computers and Cognition, A New Foundation for Design, A Response to the reviews, Artificial Intelligence 31 (1987) 250-261.

[OWI] Železnikar, A.P., On the Way to Information (Na poti k informaciji), Informatica 11 (1987) 1, 4-18.

[IDI] Železnikar, A.P., Information Determinations I, Informatica 11 (1987) 2, 3-17 and Cybernetica 31 (1988) 3, 181-213.

[ID2] Železnikar, A.P., Information Determination II, Informatica 11 (1987) 4, 8-25 and Cybernetica 32 (1989) 1, xx-xx.

[IL1] Železnikar, A.P., Informational Logic I, Informatica 12 (1988) 3, 26-38.

[IL2] Železnikar, A.P., Informational Logic II, Informatica 12 (1988) 4, 3-20.

[IL3] Železnikar, A.P., Informational Logic III, Informatica 13 (1989) 1, 25-42.

[IL4] Železnikar, A.P., Informational Logic IV, Informatica 13 (1989) 2, 6-23.

MOGUĆNOSTI PROCEDURALNOG PROGRAMIRANJA U PROGRAMSKOM JEZIKU PROLOG

INFORMATICA 3/89

Keywords: procedural programming, Prolog, matrix multiplication

Iztok Z. Race,
Mašinski fakultet, Beograd

SAŽETAK

Programski jezik PROLOG kao deklarativni programski jezik se bitno razlikuje od proceduralnih programskih jezika, jer prvenstveno opisuje posmatrani problem, a ne način kako se rešava neki problem. U skladu sa tako koncipiranim programskim jezikom, bitno se razlikuju programi pisani u PROLOG-jeziku od programa pisanih u proceduralnim jezicima. Zbog toga se mora govoriti i o drugačijem načinu razmišljanja programera prilikom izrade programa u programskom jeziku PROLOG. U radu je dato moguće rešenje za neke uobičajene programske strukture i strukture podataka iz proceduralnih jezika, te ilustracija mogućnosti proceduralnog programiranja u PROLOG-jeziku na primeru množenja matrica.

1. PROLOG KAO DEKLARATIVNI PROGRAMSKI JEZIK

Poslednjih godina, naglim razvojem veštačke inteligencije i njenom primenom u raznorodnim ljudskim aktivnostima, za programski jezik PROLOG kao oruđe u ovoj oblasti, izuzetno raste interesovanje.

PROLOG-jezik je nastao iz ideje da se predikatska logika posmatra kao programski jezik. Zbog toga, program u ovom jeziku opisuje relacije između pojedinih objekata, i pravila na osnovu kojih se iz postojećih relacija mogu dokazati nove. To je bitna razlika koja odvaja ovaj jezik od proceduralnih jezika kao što su npr. FORTRAN, BASIC ili PASCAL. U proceduralnim jezicima se program sastoji iz niza koraka u kojima se opisuje kako da računar izvrši određena izračunavanja. U PROLOG-jeziku je program opis problema koji se posmatra, dok se sam način na koji se problem rešava direktno ne opisuje. Drugim rečima, proceduralni programski jezici opisuju **kako** se rešava pojedini problem, dok deklarativni programski jezici opisuju šta je posmatrani problem.

PROLOG-jezik se od proceduralnih jezika razlikuje u nizu osobina. Promenljive u PROLOG-jeziku dobijaju vrednost samo u toku pokušaja zadovoljenja određene rečenice PROLOG-jezika. Njihova vrednost je trenutna i vezana je za trajanje razmatranja date rečenice. Na taj način promenljiva ne predstavlja striktno određenu memorijsku lokaciju. Sa druge strane, proceduralni jezici imaju velik broj sličnih programskih struktura ili struktura podataka koje ne postoje u PROLOG-jeziku poput if-then-else strukture, petlji ili nizova. Nasuprot tome PROLOG-jezik ima svoje osnovne mehanizme koji ga izdvajaju od ostalih jezika kao što su automatski backtracking i cut-predikat. Ovi mehanizmi omogućavaju da se pojedini problemi u programskom jeziku PROLOG reše na drugačiji i bolji način, što sa svoje strane može predstavljati teškoću prilikom pisanja programa u ovom jeziku, pogotovu ako se prilikom programiranja koriste načini razmišljanja iz drugih programskih jezika.

Ponekad je, u kompleksnijim problemima koji se rešavaju u PROLOG-jeziku, potrebno izvršiti određena izračunavanja, za koja su proceduralni jezici daleko efikasniji. Od načina na koji se problem postavi u PROLOG-jeziku vrlo često zavisi i brzina, a ponekad i ukupan uspeh datog rešenja. U radu se ukazuje na neke od mogućih pristupa tom problemu.

2. PREDSTAVLJANJE NEKIH STRUKTURA PODATAKA U PROGRAMSKOM JEZIKU PROLOG

Promenljive. U mnogim programskim jezicima promenljive predstavljaju određene memorijske lokacije, čiji sadržaji ostaju nepromenjeni sve dok se ne izračuna neka druga vrednost i memoriše kao novi sadržaj posmatrane memorijske promenljive. Za razliku od tog pristupa, promenljive u programskom jeziku PROLOG imaju naglašeno lokalan karakter, tj. one su aktuelne samo u okviru rečenice u kojoj se spominju. Takođe je karakteristično da svaki pokušaj izmene vrednosti već postavljene memorijske promenljive, u ovom programskom jeziku završava neuspehom.

Zbog ekstremne lokalnosti promenljivih u PROLOG-jeziku, prenos parametara se vrši na jedan od sledeća dva načina:

- funkcijskim pozivom

- upisivanjem vrednosti promenljive kao činjenice.

Primer funkcijskog poziva u PROLOG-jeziku bi bio:

```
a1(P1,P2,...).
a1(P1,P2,...):-...,a2(P1,P2,...),...
```

čime se vrednosti promenljivih P1,P2,... u predikatu a1, postavljaju na vrednosti promenljivih u predikatu a2, pa se na taj način i obavlja prenos parametara.

Upisivanje vrednosti promenljive kao činjenice u bazu podataka PROLOG-jezika se vrši pomoću sistemskog predikata asserta ili assertz. Tako bi npr. rečenica:

```
assertz(vrednost(a2,1000)).
```

u bazu podataka (na njen kraj) upisala PROLOG-činjenicu vrednost(a2,1000). Na taj način se, formiranjem ovakve strukture u bazi podataka PROLOG-jezika, može predstaviti promenljiva a2 kojoj je pridružena vrednost 1000, odnosno relacija **vrednost** koja se uspostavlja između atoma (imena promenljive) a2 i atoma (sadržaja promenljive) 1000. Promena ove vrednosti, na recimo 500, se vrši, pomoću sistemskih predikata retract i assertz, odnosno tako što se prvo obriše stara vrednost, a zatim upiše nova, odnosno, u PROLOG-jeziku:

```
retract(vrednost(a2,_)),
assertz(vrednost(a2,500)).
```

To, dinamički gledano odgovara činjenici da atomi a2 i 1000 više nisu u relaciji vrednost, i da je ta relacija uspostavljena između atoma a2 i 500. Tako se pomoću relacija može predstaviti izmena sadržaja memorijske promenljive u proceduralnim jezicima. Pri tome se mora naglasiti, da za razliku od situacije kod proceduralnih jezika, atom a2 (označava ime promenljive) može biti u relaciji sa više atoma koji označavaju vrednosti istovremeno, što može korisno poslužiti pri rešavanju nekih problema.

Nizovi. U PROLOG - jeziku nema nizova za razliku od mnogih drugih jezika. Zato se, ukoliko je potrebno određene podatke strukturirati u niz, oni unose u bazu podataka PROLOG-jezika slično promenljivama:

```
assertz(vrednost(a(1,2),550)).
```

Strukturiranje se može izvršiti i na druge slične načine.

Pokazivači. Pokazivači se u PROLOG - jeziku realizuju na taj način što se doda još jedno polje u postojeću strukturu promenljive koje ukazuje na neku drugu promenljivu. Tako strukturisana promenljiva ima oblik:

```
a(indexm,...,indexp)
```

gde promenljiva u nizu a sa indeksom indexm ima pokazivač koji pokazuje na promenljivu sa indeksom indexp.

Stekovi. Stekovi u programskom jeziku PROLOG, uglavnom se predstavljaju ili kao liste, ili kao niz činjenica.

Stek se predstavlja pomoću liste na taj način što se oformi struktura

```
stek([vrh,...,dno])
```

gde su elementi liste, redom elementi u steku, od elementa na vrhu steka (označenog sa vrh) do elementa na dnu steka (označenog sa dno). Dodavanje novog elementa na stek se vrši zadovoljenjem predikata push, koji može imati sledeći oblik:

```
push(Element,Stek,[Element|Stek]).
```

dok se izbacivanje elementa sa vrha steka može realizovati na sledeći način u PROLOG-jeziku:

```
pop(_,[],[]):-write('Stack underflow').
pop(Element,[Element|Stek],Stek).
```

Prva rečenica kojom je opisan predikat pop odgovara stanju kada se pokušava da izbaci element iz praznog steka, dok druga rečenica odgovara izbacivanju elementa u regularnom slučaju.

Stek se može predstaviti i kao niz činjenica u PROLOG-jeziku, oblika:

```
stek(vrh).
.........
stek(elementm).
.........
stek(dno).
```

Pri tome se ubacivanje elemenata u stek vrši zadovoljavanjem predikata push koji ima sledeći oblik:

```
push(Element,Stek):-
     Struktura=..[Stek,Element],
     asserta(Struktura).
```

Zadovoljavanjem predikata pop se vrši izbacivanje elemenata iz steka:

```
pop(Element,Stek):-
     Struktura=..[Stek,Element],
     pop1(Struktura).

pop1(Struktura):-
     retract(Struktura),!.
pop1(Struktura):-
     write('Stack underflow').
```

Zadovoljavanje predikata pop se vrši na taj način što se prvo zadovolji univ-predikat i tako formira struktura čiji je funktor sadržaj promenljive Stek (koja za vrednost ima ime steka) a jedini argument nepostavljena promenljiva Element. Zatim se zadovoljava predikat pop1. Ukoliko postoji u bazi podataka PROLOG-jezika struktura sa gore određenim funktorom (prva takva na vrhu) se izbacuje iz baze (i iz steka). Time se, ujedno i postavlja promenljiva Element u strukturi i predikatu pop. U slučaju da u bazi podataka takva struktura ne postoji, javlja se odgovarajuća poruka.

Niska (queue). Niske se mogu implementirati u PROLOG-jeziku kao liste, pa se

formira sledeća struktura:

```
niska([početak,...,kraj])
```

međutim takvo rešenje ima svojih loših strana, koje se uočavaju prilikom umetanja novih elemenata na kraj niske. Da bi postavili element iza elementa koji je na kraju liste, moramo zadovoljiti predikat insert koji može imati sledeći oblik:

```
insert(Element,[],[Element]).
insert(Element,[G|R],[G|R1]):-
     insert(Element,R,R1).
```

Izbacivanje elemenata iz liste u takvom rešenju je daleko jednostavnije obzirom da je element koji se izbacuje iz niske prvi element liste kojom je niska predstavljena. Tako bi predikat delete kojim obezbeđujemo tu funkciju mogao da ima oblik:

```
delete(_,[],[]):-
     write('Queue underflow'),!.
delete(Početak,[Početak|R],R).
```

Mnogo efikasnije rešenje se postiže upotrebom sistemskog predikata assertz, u kojem se niska predstavlja u obliku činjenica u PROLOG-jeziku tipa:

```
niska(početak).
.............
niska(elementm).
.............
niska(kraj).
```

Predikati insert i delete koji nam služe za manipulaciju sa niskom bi u ovom slučaju imali oblik:

```
insert(Element,Niska):-
     Struktura=..[Niska,Element],
     assertz(Struktura).

delete(Element,Niska):-
     Struktura=..[Niska,Element],
     delete1(Struktura).

delete1(Struktura):-
     retract(Struktura),!.
delete1(Struktura):-
     write('Queue underflow').
```

Drveta i grafovi se mogu predstaviti na razne načine koji variraju od rešenja da se celo drvo (ili graf) predstavi kao lista do rešenja da se prvo čvorovi drveta (grafa) predstave u obliku činjenica u PROLOG-jeziku, a potom, takođe i veze koje postoje među tim čvorovima. Tako se graf može predstaviti sledećom strukturom u PROLOG-jeziku:

```
graf([[čvor1,[čvor11,...,čvor1k]],...,
                [[čvorn,[čvorn1,...,čvornm]]])
```

gde su čvor1,...,čvorn imena čvorova u grafu uz koje je pridružena lista imena čvorova sa kojima je svaki od tih čvorova u vezi. Graf se može predstaviti i na sledeći način:

```
graf(čvor1,[čvor11,...,čvor1k]).
.............................
graf(čvorn,[čvorn1,...,čvornm]).
```

ili:

```
graf(čvor1).
...........
graf(čvorn).
graf(čvor1,čvor11).
...................
graf(čvorn,čvornm).
```

gde se činjenice sa funktorom graf i jednim argumentom odnose na čvorove grafa, a one činjenice sa istim funktorom i dva argumenta na veze koje postoje u grafu. Ovo su samo neke od varijanata predstavljanja grafa (drveta) u PROLOG-jeziku. Svaka od varijanata mora da bude prvenstveno prilagođena problemu. U zavisnosti od izabrane varijante su i predikati koji omogućuju manipulaciju tako predstavljenim grafovima ili drvetima.

3. PREDSTAVLJANJE NEKIH UOBIČAJENIH PROGRAMSKIH STRUKTURA U PROLOG JEZIKU

if-then-else (if-then) struktura se javlja u formi:

```
if uslov then naredba1 else naredba2
```

ili

```
if uslov then naredba.
```

Prvi oblik se u PROLOG-jeziku implementira kao:

```
if_then_else(Uslov,Naredba1,Naredba2):-
     Uslov,!,
     Naredba1.
if_then_else(Uslov,Naredba1,Naredba2):-
     Naredba2.
```

ili:

```
if_then_else(Uslov,Naredba1,Naredba2):-
     ((Uslov,!,Naredba1);Naredba2).
```

Ovde su Uslov, Naredba1 i Naredba2 promenljive koje se postavljaju na imena predikata koji opisuju traženi uslov i odgovarajuće naredbe.

U drugom obliku (if-then) se javljaju problemi u slučaju neispunjenja uslova. Tada se mora obratiti pažnja na to da predikat bude zadovoljen i u slučaju da postavljeni uslov nije ispunjen, pošto bi pad predikata mogao da prouzroči i neizvršenje programa. Tako bi odgovarajući predikat imao oblik:

```
if_then(Uslov,Naredba):-
     Uslov,!,
     Naredba.
if_then.
```

ili:

```
if_then(Uslov,Naredba):-
     ((Uslov,!,Naredba);true).
```

Očito je da se ovakve strukture mogu dalje usložnjavati i da se na sličan način može predstaviti i naredba višestrukog grananja (case naredba u PASCAL-jeziku).

while-do petlja ima oblik:

```
while uslov do naredba.
```

Ukoliko petlja nije beskonačna, uslov je funkcija argumenata koji menjaju svoje vrednosti unutar ciklusa tokom izvršenja naredbe. Jedan od mogućih pristupa je da se ova programska struktura u PROLOG-jeziku predstavi rekurzivnim rešenjem:

```
while_do(Uslov,Naredba):-
     not(Uslov),!.
while_do(Uslov,Naredba):-
     Naredba,
     while_do(Uslov,Naredba).
```

Ovo rešenje ima svoje nedostatke u rekurzivnom pristupu rešenju programskog ciklusa u PROLOG-jeziku i memorisanju parametara u rekurzivnom pozivu koje je potrebno zbog obezbeđenja mehanizma backtracking-a.

Rešenje može da ima i sledeći oblik u kome se koristi sistemski predikat repeat:

```
while_do(Uslov,Naredba):-
     Uslov,
     repeat_while(Naredba,Uslov).
while_do(_,_).

repeat_while(Naredba,Uslov):-
     repeat,
     Naredba,
     not(Uslov).
```

U ovom rešenju je izbegnuto nepotrebno memorisanje parametara u rekurzivnom pozivu koje je glavni nedostatak prethodno navedenog rešenja.

Komponovane naredbe visokog nivoa u velikim sistemima se javljaju kao poseban problem prilikom izgradnje aplikacija u PROLOG-jeziku. Naime, vrlo često se dešava da je potrebno po određenom redosledu zadovoljiti neke predikate, bez potrebe memorisanja puta kojim se prošlo (radi eventualnog backtracking-a). U ovakvoj situaciji predikati u PROLOG-jeziku se ne posmatraju u svom deklarativnom, već prvenstveno u proceduralnom značenju. Tako se predikati posmatraju kao procedure koje obavljaju određenu akciju. Memorisanje puta kojim se prošlo se prekida sistemskim predikatom fail. Parametri koji se prenose između tako posmatranih procedura se mogu upisivati u bazu podataka PROLOG-jezika pomoću sistemskih predikata asserta i assertz, kao i čitati i brisati pomoću predikata retract. Primera radi komponovana naredba:

```
begin naredba1;naredba2;...;naredbak end
```

bi bila realizovana zadovoljenjem predikata komp_naredba:

```
komp_naredba:-
     naredba1,
     fail.
komp_naredba:-
     naredba2,
     fail.
.............
komp_naredba:-
     naredbak.
```

Na ovaj način je moguće, vodeći računa o tome kada je potreban pojedini predikat u svom proceduralnom, a kada u deklarativnom značenju, izgraditi dosta velike sisteme koji imaju i proceduralne delove.

4. PRIMER PROCEDURALNOG PROGRAMIRANJA U PROLOG-JEZIKU - MNOŽENJE MATRICA

Klasičan proceduralni problem je izrada programa za množenje matrica. Neka su matrice a i b date u obliku činjenica u PROLOG-jeziku, na način koji je naveden u tekstu za predstavljanje nizova:

```
vrednost(a(1,1),Va11).
.....................
vrednost(a(1,M),Va1M).
vrednost(a(2,1),Va21).
.....................
.....................
vrednost(a(N,M),VaNM).
vrednost(b(1,1),Vb11).
.....................
vrednost(b(1,K),Vb1K).
vrednost(b(2,1),Vb21).
.....................
.....................
vrednost(b(M,K),VbMK).
```

i neka su promenljive N, M, K i Va11,...,VaNM,Vb11,...,VbMK postavljene na konkretne numeričke vrednosti. Program za množenje tako zadatih matrica i formiranje odgovarajućih činjenica o matrici-rezultatu u bazi podataka ima sledeći oblik:

```
mnozenje_matrica(A,B,C):-
     Struktura1=..[A,A1,A2],
     Struktura2=..[B,A2,B2],
     Struktura3=..[C,A1,B2],
     vrednost(Struktura1,V1),
     vrednost(Struktura2,V2),
     element_prod(Struktura3,V1,V2,V),
     assertz(vrednost(Struktura3,V)),
     fail.
mnozenje_matrica(_,_,_).
element_prod(Struktura3,V1,V2,V):-
     retract(vrednost(Struktura3,V3)),!,
     V is V3+V1*V2.
element_prod(Struktura3,V1,V2,V):-
     V is V1*V2.
```

Pozivom mnozenje_matrica(a,b,c) u PROLOG bazi podataka se formiraju strukture vrednost koje odgovaraju matrici c koja je rezultat množenja matrica a i b. Koristeći univ-predikat i ne postavljajući promenljive A1,A2 i B2 formiraju se odgovarajuće strukture koje odgovaraju onim elementima matrice koji se trenutno množe

(Struktura1 i Struktura2) i koji se trenutno izračunava (Struktura3). Kao elementi ovakvih struktura, nalaze se promenljive koje dobijaju vrednosti koje odgovaraju indeksima matrice. Pomoću tih promenljivih se uspostavljaju i relacije unutar baze podataka PROLOG-jezika i time obezbeđuje množenje odgovarajućih članova matrica i upisivanje rezultata na odgovarajuće mesto. Zadovoljavanjem predikata vrednost dobijaju se konkretne vrednosti za elemente matrica sa konkretnim indeksima. Zadovoljavanjem predikata element_prod se sračunava proizvod vrednosti dva odgovarajuća elementa matrica i dodaje na ukupan međuzbir pri izračunavanju elemenata u matrici-rezultatu. Tako izračunata vrednost (međuzbir ili konačni element matrice-proizvoda) se pomoću sistemskog predikata assertz upisuje u bazu podataka PROLOG-jezika. Nakon ovoga predikat mnozenje_matrica pada (sistemski predikat fail) i pokušava da se ponovo zadovolji postavljajući nove vrednosti promenljivih A1,A2 i B2. One su određene činjenicama vrednost u bazi podataka PROLOG-jezika. Nastavljajući tako, iscrpljujući skup svih proizvoda elemenata matrica definisanih činjenicom vrednost koji zadovoljavaju relacije određene u definisanim strukturama, dobija se konačan rezultat. Druga rečenica koja definiše predikat mnozenje_matrica služi isključivo da bi predikat na kraju bio zadovoljen.

Interesantno je primetiti da činjenice o vrednostima elemenata matrice u bazi podataka ne moraju uopšte biti poređane ni po vrstama ni po kolonama, i da matrice ne moraju biti kompletne (mogu nedostajati pojedini elementi) da bi bilo izračunato ono što se u tom slučaju može izračunati u množenju takvih matrica.

Važno je takođe primetiti da se korišćenjem sistemskog predikata fail može postići da se neprekidnim ponovnim zadovoljavanjem željenih predikata ponavlja određena procedura (posmatrajući proceduralno značenje tih predikata). Izmenom nekih elemenata (upisivanjem ili brisanjem u bazi podataka PROLOG-jezika pomoću asserta,assertz i retract) može se različito uticati na trajanje tih ponavljanja. Na taj način se mogu dati rešenja u programskim ciklusima, a time se i izbegava odgovarajuće rekurzivno rešenje koje je nepogodno zbog memorisanja pređenog puta.

5. UMESTO ZAKLJUČKA

Ovaj rad ima dve namene. Prva je da programerima koji su početnici u pisanju programa u PROLOG-jeziku omogući da svoje proceduralne programe lako prevedu u PROLOG-jezik. Ipak se mora naglasiti da tako prevedeni programi nisu najsretnije rešenje. Bez obzira na to dosta velik broj programera razmišlja proceduralno prilikom izrade svojih programa. Na taj način se dolazi i do druge namene rada. Naime, radom se želelo pokazati da se iskusan programer u drugim programskim jezicima, prilikom programiranja u PROLOG-jeziku mora osloboditi nekih ranije stečenih dogmi u gledanju na pojedine probleme. Lep primer za to je množenje matrica. Uobičajena navika iz drugih programskih jezika je da procedura bude tačno završena (da ne "padne" tokom izvršavanja). U većini programskih jezika se takva mogućnost "pada" ni ne razmatra. "Side"-efekti tokom izvršenja procedure se uglavnom posmatraju kao nepovoljna osobina, pa se ili izbegavaju, ili upotrebljavaju u ograničenim količinama. Otuda je često mišljenje koje ide linijom manjeg otpora da se "side"-efekat izbacuje iz razmišljanja tokom izrade procedura. O "padu" procedure tokom njenog izvršenja se u početku pisanja programa u PROLOG-jeziku uglavnom ne razmišlja. Rešenje koje je dato za problem množenja matrica ilustruje neprestano "padanje" predikata mnozenje_matrica (posmatranog kao procedura) kao i "side"-efekte dobijene tom prilikom (upis novoizračunatih međuzbirova u bazu podataka PROLOG-jezika pomoću sistemskih predikata assertz i retract).

Usvajanjem ovakvog načina razmišljanja prilikom pisanja programa u PROLOG-jeziku bitno se doprinosi poboljšanju stila pisanja tih programa, a i efikasnosti njihovog izvršenja.

6. LITERATURA

1. Bratko, Ivan, Inteligentni informacijski sistemi, Univerza Edvarda Kardelja v Ljubljani, Fakulteta za elektrotehniko, Ljubljana, 1984
2. Clocksin, W. F., Melish, C. S., Programming in Prolog, Springer Verlag, Berlin Heidelberg New York, 1981
3. Munakata, Toshinori, Procedurally Oriented Programming Techniques in Prolog, IEEE Expert, Summer 1986, str 41-47
4. Race, Iztok Z., Programska implementacija ekspertnog sistema za dijagnostiku u PROLOG-jeziku, magistarski rad, Univerzitet u Beogradu, Prirodno matematički fakultet, Matematički fakultet, Beograd, 1988
5. Sterling, Leon, Shapiro, Ehud, The Art of Prolog: Advanced Programming Techniques, The MIT Press, Cambridge, Massachusetts, London, England, 1986

MERJENJE PARALELNOSTI ALGORITMOV .

INFORMATICA 3/89

Borut Jereb
Ljubo Pipan*
Institut J. Stefan
Fakulteta za elektrotehniko in računalništvo*, Ljubljana

Keywords: parallel algorithms, measuring of parallelism, task graph, simultaneous degree, coupled degree

Paralelne algoritme lahko predstavimo z grafom opravil. Iz grafa opravil je možno enostavno razbrati stopnjo sočasnosti in stopnjo povezanosti algoritma, ki ga graf predstavlja. Ti stopnji predstavljata pomembni značilnosti paralelnih algoritmov.

V tem članku je podan predlog za izračun mere sočasnosti in mere povezanosti. Opisane so tudi nekatere značilnosti obeh mer.

Izračun mer je neodvisen od arhitekture, na katerem bo tekel algoritem; sami meri pa lahko predstavljata velikost vložka, ki je potreben za sprogramiranje algoritma.

MEASURING THE PARALLELISM

Parallel algorithms could be represented by the task graph, which is at the same time a simple representation of the simultaneousness degree and coupled degree of an algorithm, represented by the graph. The above mentioned degrees are most important features of the parallel algorithm.

This study suggests a method for calculating the simultaneousness degree and the coupled degree. Included are some of their main characteristics.

The calculation of the degrees does not depend on the architecture within which the algorithm will operate; the degrees themselves though represent the size of the insert required for the algorithm to be programmed.

UVOD

Ko začnejo človeka zanimati paralelne arhitekture, paralelno procesiranje,... in sploh vse, kar naj bi bilo paralelnost, vzame v roko literaturo in prične s študiranjem le te. Literature v obliki knjig in člankov je ogromno, saj je to že vrsto let popularna tema, ki je še vedno do neke mere ovita v plašč misterioznosti in zato tem bolj buri duhove. Pri prebiranju literature si ustvarjamo svojo predstavo o stvareh, ki so predmet avtorjevih razmišljanj. Eni takšno, drugi drugačno - pač odvisno od osebe.

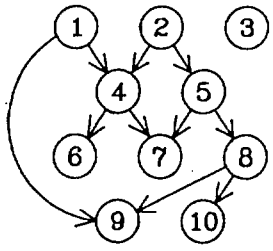
No vrnimo se nazaj k paralelnosti. V literaturi, ki obravnava paralelizem je ničkolikokrat omenjena SOČASNOST in medsebojna odvisnost ali POVEZANOST v zvezi z opravili, procesi, instrukcijami. Ne mislim se spuščati v definicijo(e) paralelnosti, saj jih je veliko, če ne celo preveč; vsakemu pa je intuitivno jasno kaj to je. Podobno je s sočasnostjo in povezanostjo. In kaj je z mero za sočasnost in mero za povezanost? V tem članku bom skušal podat ti meri. Naj se imenujeta stopnja sočasnosti in stopnja povezanosti.

Kakor je ob praznih kozarcih brezpredmetno govoriti o kvaliteti vina, ki naj bi bilo v kozarcih, tako je brezpredmetno govoriti o bolj ali manj paralelnih algoritmih. V tehniki je potrebno pojme oceniti. Ocena pojma je dobljena po točno določenem algoritmu in je v veliki večini primerov predstavljena s števili iz intervala, ki jih ta ocena lahko zavzame.

Postavlja se vprašanje, kako oceniti paralelizem, ki smo ga izbrali oziroma določili v nekem algoritmu. Sem prepričanja, da mera za sočasnost in mera za povezanost programskih modulov, dajeta dobro osnovo za oceno paralelizma v poljubnem algoritmu. Pri tem sta ti meri neodvisni od arhitekture na kateri se naj bi izvajal algoritem in s tem daje splošno primerljive rezultate.

PREDSTAVITEV PARALELNEGA ALGORITMA

Najprej bo opisan način predstavitve paralelnega algoritma (način določevanja paralelnih modulov in določevanje njihove medsebojne - večkrat podatkovne odvisnosti - naj nas ne zanima). Paralelni algoritem naj bo predstavljen z grafom. Takšna predstavitev je v literaturi splošno uporabljana.[1,2,3,4,6]



Slika 1 Predstavitev paralelnega algoritma z grafom opravil.

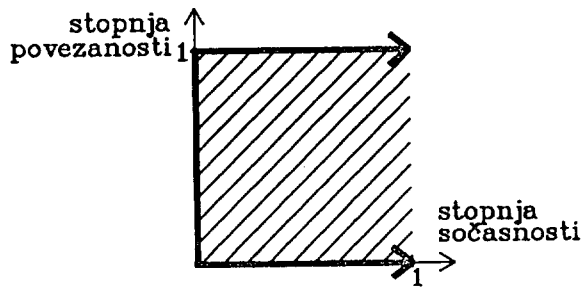
Ker je paralelni algoritem predstavljen z grafom, si oglejmo lastnosti grafa tako, da jih bomo primerjali z lastnostmi paralelnih algoritmov. Vozlišče grafa predstavlja programski modul. Vsi programski moduli, pri katerih je mišljeno sočasno izvajanje, imajo isti nivo. Nivo je utež vozlišča. Pri tem velja, da ležijo na nivoju ena vsi tisti programski moduli, ki se izvajajo sočasno in so neodvisni med seboj in neodvisni od ostalih programskih modulov. Sočasno izvajajoči programski moduli, ki so med seboj neodvisni in odvisni od programskih modulov na nivoju ena, ležijo na nivoju dva. Nivo tri predstavlja množico, med seboj neodvisnih programskih modulov, ki so odvisni od prograskih modulov na obeh nižjih nivojih in se izvajajo sočasno; itd. Odvisnost med programskimi moduli je v grafu predstavljena z usmerjenimi povezavami med vozlišči grafa. Povezava poteka vedno v smeri od vozlišča, ki leži na nižjem nivoju, k vozlišču na višjem nivoju. Pri tem lahko ostanejo nekatera vozlišča v grafu nepovezana. Graf je acikličen, saj lahko ciklično izvajanje nekega programskega modula predstavimo s podvajanjem tega programskega modula. Definirajmo še višino grafa, ki naj bo enaka največjemu nivoju, ki ga ima katerokoli od vozlišč grafa.

Rekurzivno izvajanje programskih modulov rešimo s podvajanjem vozlišča, ki predstavlja rekurzivni programski modul.

Za primer predstavitve nekega paralelnega algoritma z grafom, si oglejmo Sliko 1, ki predstavlja algoritem, razdeljen na deset modulov. Pri tem je možno sočasno izvajanje modulov, predstavljenih z vozlišči 1, 2 in 3, ki ležijo na nivoju ena. Nivo ena vsebuje tudi nepovezan programski modul, oziroma izolirano točko. Na nivoju dva sta dva medsebojno neodvisna sočasno tekoča programska modula, ki sta odvisna od modulov prejšnjega nivoja. Ker obstajata še dva nivoja, je višina grafa enaka štiri.

DEFINICIJI MER

Najprej bom opisal vrednosti, ki jih ti dve meri lahko zavzameta. Obe naj bosta iz množice racionalnih števil na intervalu med nič in ena. In kako dosežemo skrajni točki pri obeh merah? Mera za sočasnost naj bo enaka nič, ko je algoritem popolnoma sekvenčen (zaporeden). Vrednost ena pa naj zavzame takrat, ko je možno razbiti algoritem na neskončno modulov, ki se bodo izvajali v istem časovnem intervalu, oziroma trenutku. Mera za povezanost naj bo enaka nič, ko so vsi programski moduli med seboj neodvisni in enaka ena, ko je vsak programski modul odvisen od vseh predhodno izvedenih programskih modulov, pri popolnoma sekvenčnem programu.



Slika 2 Področje v katerem je definirana dvojica (stopnja sočasnosti, stopnja povezanosti).

Če so lastnosti paralelnih algoritmov predstavljene z dvema številoma, lahko dvojica (stopnja sočasnosti, stopnja povezanosti) zavzame, glede na vse dosedaj povedano, vse vrednosti, ki ga določa šrafirano področje in poudarjena črta na sliki 2.

Definirajmo:

1. Stopnja sočasnosti:

$$CD = \frac{n - TH}{n}$$

pri čemer je $n > 1$ in predstavlja število vozlišč grafa. TH predstavlja višino drevesa.

2. Stopnja povezanosti

$$SD = \frac{EN}{\frac{n(n-1)}{2}}$$

pri čemer je $n > 1$ in predstavlja število vozlišč grafa. EN predstavlja število povezav v grafu.

Iz teh dveh definicij sledi nekaj trditev:

1. Maksimalno stopnjo sočasnosti dobimo v limiti, ko smo problem razdelili na neskončno programskih modulov, ki se izvajajo sočasno. Limita stopnje sočasnosti je takrat enaka 1.
2. Minimalno stopnjo sočasnosti dobimo pri popolnoma sekvenčno organiziranem algoritmu in je enaka 0.
3. Pri $SD = 0$ je možno vse module paralelnega algoritma izvajati obenem. Velja tudi obratno.

4. Pri $CD = 0$ je $\frac{2}{n} \leq SD(n) \leq 1$.
5. Pri dani stopnji sočasnosti je spodnja meja za mero povezanosti enaka $\frac{TH-1}{\frac{(n-1)n}{2}}$.

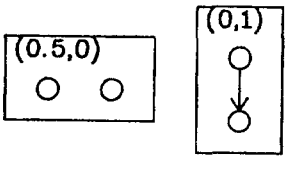
Z definicijo neke nove mere na osnovi pravkar definiranih stopenj bi izgubili na enostavnosti in jasnosti predstavitve s tema dvema, za paralelizem pomembnima lastnostima paralelnih algoritmov.

LASTNOSTI DVOJICE (CD, SD)

Vrednosti obeh mer dajeta slutiti obliko grafa, ki predstavlja algoritem. Potrebno je tudi povedati, da smo pri tem izpustili dodatno utežitev vozlišč, ki bi predstavljala čas izvajanja. Čas bi lahko bil absoluten, izražen v časovnih enotah ali relativen glede na čas izvajanja nekega opravila v sistemu. Takšna merjenja so v literaturi obdelana.[2] So zahtevnejša, ter nemalokrat zavisijo od podatkov in ostalih težko predvidljivih situacij, ki nastanejo med izvajanjem programa. Vrednosti mer za algoritem, ki ga predstavlja eno samo vozlišče ni definirano.

Pred točnim definiranjem obeh mer smo omejili področje, ki ga ti dve meri lahko zavzameta. Toda nekatere kombinacije CD in SD so nesmiselne. Na osnovi trditev in primerov si oglejmo vrednosti, ki jih lahko zavzame dvojica (CD, SD). Pri tem bomo pregledali vse možne vrednosti za CD in SD, pri razbitju algoritma na n modulov. Točke, ki bodo predstavljale mejne vrednosti, naj bodo povezane z daljicami.

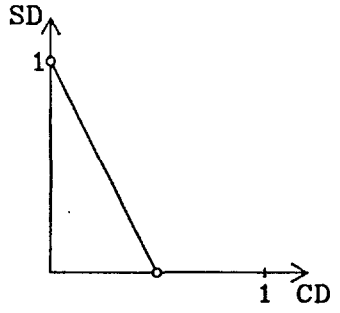
Ker vrednosti mer za algoritem, ki ga predstavlja eno samo vozlišče, ni definirano, si najprej oglejmo primer, pri katerem smo algoritem razbili na dva modula. Slika 3 predstavlja vse možne načine izvajanja algoritma z dvema moduloma.



Slika 3 Vsi možni načini izvajanja paralelnega algoritma z dvema moduloma.

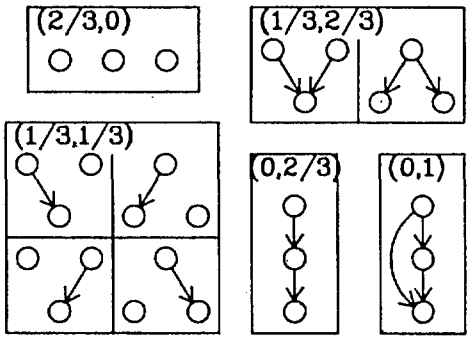
Vidimo, da sta možna le dva načina: popolnoma paralelen in popolnoma sekvenčen način izvajanja. Slika 4 predstavlja vrednosti, ki jih dvojica (CD, SD) lahko zavzame v $CD - SD$ diagramu. Če mejni točki pove- žemo, dobimo med njima daljico.

Iz primera je razvidno, da je pri popolnoma paralelnem načinu izvajanja algoritma mera za sočasnost enaka 0.5 in mera za povezanost enaka 0. Morda je na prvi pogled presenetljivo dejstvo, da ima 'popolnoma' paralelen algoritem stopnjo so asnosti enako le 0.5. Toda, če smo algoritem razbili na dva modula, je to čisto nekaj drugega kot, če smo ta isti algoritem uspeli razbiti na npr. 20000 med sabo popolnoma neodvisnih modulov. Zgoraj definirana stopnja sočasnosti upošteva to razliko.



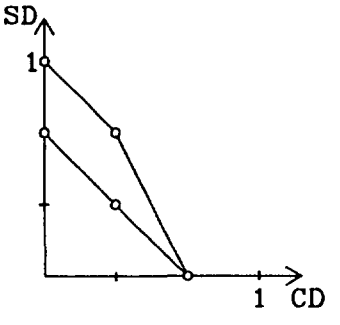
Slika 4 Daljica, ki povezuje obe možni vrednosti (CD,SD) paralelnega algoritma z dvema moduloma.

Kakšne so vse možnosti izvajanja in katere vrednosti lahko zavzame dvojica (CD, SD) pri razbitju algoritma v tri module, podajata sliki 5 in 6.



Slika 5 Vsi možni načini izvajanja paralelnega algoritma s tremi moduli

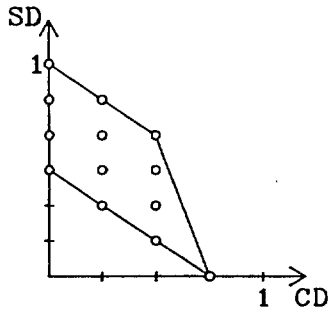
Če povežemo med seboj mejne točke, kot je to prikazano na sliki 6, dobimo nek lik.



Slika 6 Lik, ki nastane s povezovanjem vrednosti (CD,SD) paralelnega algoritma s tremi moduli.

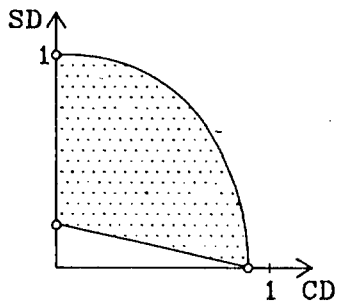
Slika 7 prikazuje možne vrednosti dvojic (CD,SD) v $CD - SD$ diagramu pri razbitju algoritma na štiri module. Lik, ki ga dobimo s povezavo vseh mejnih

točk, spominja na lik, ki smo ga dobili v prejšnem primeru.



Slika 7 Lik, ki nastane s povezovanjem vrednosti (CD,SD) paralelnega algoritma s štirimi moduli.

Slika 8 prikazuje možne vrednosti dvojic (CD,SD) pri razbitju algoritma na večje število modulov (deset). Vse možne vrednosti na sliki niso narisane, oblika lika pa približno podaja območje, v katerem se te vrednosti lahko nahajajo. Tudi z rastočim številom programskih modulov oblika lika ostaja podobna prejšnima dvema.



Slika 8 Približen lik, ki nastane s povezovanjem vrednosti (CD,SD) paralelnega algoritma z desetimi moduli.

Na osnovi primerov smo se prepričali, da so mnoge kombinacije vrednosti stopnje sočasnosti in stopnje povezanosti nesmiselne.

Pravtako je iz primerov razvidno, da je oblika lika, ki ga dobimo s povezovanjem mejnih točk, podobna od primera do primera. Še posebej je razlika, ki nastopi pri različnem številu programskih modulov zelo majhna, ko sta ti števili veliki. Pri tem je treba izvzeti enostavni primer z dvema programskima moduloma, pri katerem se lik poenostavi v daljico.

Značilnost vseh likov je to, da imajo na CD osi (takrat je $SD = 0$) le eno točko. Ta točka z večjim številom programskih modulov potuje od $CD = \frac{1}{2}$ do $CD = 1$; vendar slednje točke nikoli ne doseže.

Od tu gre meja po daljici, ki vsebuje spodnje mejne točke povezanosti, do točke na SD osi. Z večanjem števila programskih modulov, se najnižja točka na SD osi od vrednosti $SD = 1$ približuje vrednosti $SD = 0$, ki jo pa nikoli ne doseže.

Vsi liki vsebujejo točko $SD = 1$ pri $CD = 0$.

Za vse like je tudi značilno, da so zgornje mejne točke povezanosti nad premico $SD = 1 - CD$; in to tem bolj, čim večje je število programskih modulov. Pri vseh $CD > 0$ mejne točke nikoli ne dosežejo vrednosti $SD = 1$.

SKLEP

Zgoraj opisani meri bosta morda v pomoč pri medsebojni komunikaciji, pri izražanju in pri razumevanju drugega človeka, kadar bo govor o sočasnosti in povezanosti poljubnega paralelnega algoritma. Ker opisujeta le graf opravil, ki smo ga določili na osnovi razbitja nekega algoritma, sta neodvisni od arhitekture ter časov izvajanj posameznih opravil - programskih modulov.

Algoritem za izračun obeh mer na osnovi danega grafa je zelo preprost, sam graf pa ima lastnosti, ki praktično ne predstavljajo omejitev pri opisovanju paralelnega algoritma.

LITERATURA

1. K. Hwang and F. Briggs
   Computer Architecture and Parallel Processing
   McGraw Hill, New York, 1984
2. L. Kleinrock
   Distributed Systems
   IEEE Computer 18, 11 (Nov 85), 90-103
3. B. Kruatrazchue and T. Lewis
   Grain size determination for paralel processing
   IEEE Software, jan 88, 23-32
4. J.L. Peterson and A. Silberschatz
   Operating system concepts
   Addison-Wesley pub. comp., 1984
5. D. Bajc in T. Pisanski
   Najnujneše o grafih
   Presek, letnik 12, 84-85
6. S. Jajodia, J. Liu and P. A. Ng
   A sheme of paralel processing for MIMD systems
   IEEE Software engineering, jul 83, 436-445

IMPLEMENTACIJA POIZVEDOVANJ V MREŽNIH DATOTEKAH

INFORMATICA 3/89

Keywords: grid file, file query, implementation

Viljan Mahnič

POVZETEK: V članku je opisana implementacija sistema za vzdrževanje mrežnih datotek, ki omogoča učinkovito izvajanje različnih poizvedovanj, značilnih za relacijske podatkovne baze. Uvodni razlagi osnovnih značilnosti mrežnih datotek sledi opis algoritmov za omejevanje iskalnega območja v podatkovnem prostoru in ovrednotenje selekcije nad izbrano relacijo. V relacijskih podatkovnih bazah se najpogosteje pojavljajo poizvedovanja, sestavljena iz selekcije, projekcije in stika. V članku sta opisani dve strategiji, ki omogočata realizacijo tovrstnih poizvedovanj. Na koncu sledi še opis programskega paketa, v okviru katerega je bila izvedena implementacija vseh navedenih algoritmov.

IMPLEMENTATION OF QUERIES IN GRID FILES: We describe an implementation of the grid file system which enables efficient evaluation of various relational queries. At first, the basic concepts of grid files are explained, and then the algorithms whose purpose is to restrict the search region in the data space and to compute the selection operation are described. General queries involving selection, projection, and join operations are often used to retrieve information from a relational database. Two strategies are proposed to evaluate such a query. Finally, a description of the software package in which all these algorithms were implemented is given.

1. UVOD

Mrežne datoteke (angl. grid files) predstavljajo podatkovno strukturo, ki omogoča učinkovito iskanje podatkov po večjem številu atributov. Za razliko od standardnih načinov fizične organizacije podatkov (n.pr. indeksno-sekvenčne datoteke, razpršene tabele) se v okviru mrežnih datotek vsi atributi, po katerih vršimo iskanje, obravnavajo enakovredno (simetrično), kar pomeni, da primarni ključ ni privilegiran v primerjavi z ostalimi atributi. Poleg tega pa se mrežne datoteke s sprotnim spreminjanjem svoje strukture (razcep oziroma zlivanje podatkovnih blokov in s tem povezano ažuriranje mrežnega direktorija) dinamično odzivajo na vstavljanje novih oziroma brisanje že obstoječih zapisov.

Osnovni koncepti mrežnih datotek so bili prvič opisani v članku [NHS84], ki pa je pustil odprta še nekatera vprašanja v zvezi z implementacijo skal in mrežnega direktorija kot tudi glede izbora algoritmov za razcep in zlivanje. Odgovore na ta vprašanja je poiskal Hinrichs v [Hin85a, Hin85b], ki je implementiral sistem za upravljanje mrežnih datotek in opisal možnosti njihove uporabe za hranjenje geometričnih podatkov. Schikuta [Sch86] pa je predlagal, da bi mrežne datoteke uporabili kot osnovo za implementacijo relacijskih podatkovnih baz.

Z namenom, da bi proučili možnosti uporabe mrežnih datotek na področju relacijskih podatkovnih baz, smo v Laboratoriju za programsko opremo in informatiko, ki deluje v okviru Fakultete za elektrotehniko in računalništvo, implementirali sistem za upravljanje mrežnih datotek in algoritme za izvajanje poizvedovanj na osnovi operacij relacijske algebre.

Pri implementaciji postopkov za vzdrževanje mrežnih datotek smo se v veliki meri oprli na ugotovitve iz [Hin85a, Hin85b], s to razliko, da smo namesto enostavne sheme vhodno-izhodnih vmesnikov vpeljali fleksibilnejšo, ki omogoča boljše izkoriščanje razpoložljivega prostora v glavnem pomnilniku, s tem pa zmanjšuje število dosegov na disk, potrebnih za realizacijo poizvedovanj.

Na novo pa smo implementirali postopke za omejevanje iskalnega območja in realizacijo poizvedovanj na osnovi operacij relacijske algebre.

Za boljše razumevanje članka podajamo v nadaljevanju najprej kratek opis temeljnih značilnosti mrežnih datotek. Nato sledi opis strategije za omejevanje iskalnega območja, opis algoritmov za izvajanje poizvedovanj v mrežnih datotekah in nazadnje še opis strukture programskega paketa.

2. OSNOVNE ZNAČILNOSTI MREŽNIH DATOTEK

Datoteko d smatramo kot zaporedje zapisov $z=(a_1,a_2,...,a_k)$, kjer so $a_1,a_2,...,a_k$ vrednosti posameznih atributov. Vsaka vrednost a_i pripada neki linearno urejeni domeni A_i.

Vsak zapis lahko ponazorimo kot točko v k-dimenzionalnem podatkovnem prostoru, tako da vrednosti atributov uporabimo kot koordinate v posameznih dimenzijah. Z uvedbo primerne pravokotne mreže lahko podatkovni prostor razdelimo na celice, ki imajo obliko hiperparalelepipeda, in poskrbimo, da so tiste točke (zapisi), ki se nahajajo znotraj iste celice, shranjene na disku v okviru istega bloka podatkov.

Za predstavitev mreže potrebujemo k enodimenzionalnih polj, ki jih imenujemo skale, in eno k-dimenzionalno polje, imenovano mrežni direktorij. Vsaka skala vsebuje več mej, pri čemer vsaka meja predstavlja (k-1)-dimenzionalno hiperravnino, ki deli podatkovni prostor na 2 dela. Mrežni direktorij skrbi za povezavo med mrežo v podatkovnem prostoru in dejansko organizacijo podatkov na disku. Vsaki celici v mreži ustreza en element mrežnega direktorija, ki vsebuje naslov tistega podatkovnega bloka, v katerem so shranjene vse točke (zapisi), ki se nahajajo v tej celici.

Da bi se izognili nizki zasedenosti podatkovnih blokov, lahko isti podatkovni blok hrani točke iz več različnih celic. Celice, ki jim pripada na disku isti podatkovni blok, tvorijo t.i. območje podatkovnega bloka, ki mora (zaradi algoritmov za razcep in zlivanje) imeti obliko hiperparalelepipeda.

Pri vstavljanju novega zapisa najprej s pomočjo skal določimo tisto celico v mreži, kamor se zapis preslika kot točka v podatkovnem prostoru, nato pa s pomočjo mrežnega direktorija dobimo naslov podatkovnega bloka, v katerega je treba zapis dodati. Če v podatkovnem bloku ni več prostora, se izvrši razcep podatkovnega bloka, ki lahko poteka na dva načina, odvisno od tega, kakšno je območje podatkovnega bloka.

Če območje podatkovnega bloka obsega eno samo celico, moramo v eno izmed skal dodati novo mejo, ki razdeli območje podatkovnega bloka na dva dela, in skladno s to mejo porazdeliti zapise med dva podatkovna bloka. Nova meja povzroči, da se mrežni direktorij poveča (vriniti je treba (k-1)-dimenzionalno rezino), kar zahteva prestrukturiranje mrežnega direktorija in ustrezno ažuriranje naslovov podatkovnih blokov, ki pripadajo posameznim celicam.

Če območje podatkovnega bloka obsega več celic, odpade potreba po dodajanju nove meje, saj se za razcep lahko uporabi ena izmed mej, ki že sekajo območje podatkovnega bloka. V tem primeru se mrežni direktorij ne poveča; poskrbeti moramo le za ažuriranje naslovov znotraj prvotnega območja podatkovnega bloka.

Struktura mrežnega direktorija in vsebina mrežne datoteke se dinamično prilagajata tudi pri brisanju zapisov. V primeru, ko zasedenost nekega podatkovnega bloka pade pod predpisano mejo, pride do zlitja s sosednim podatkovnim blokom, kar ima zopet za posledico ažuriranje naslovov v mrežnem direktoriju. Če meja, ki je ločevala območji obeh podatkovnih blokov, ni več potrebna, jo izločimo, pri tem pa se mrežni direktorij zmanjša (izločiti je treba (k-1)-dimenzionalno rezino).

Pri implementaciji mrežnih datotek povzroča največ težav implementacija mrežnega direktorija. Le-ta je v splošnem prevelik, da bi se lahko v celoti nahajal v glavnem pomnilniku, zato ga moramo hraniti na disku. To pa pomeni, da je prestrukturiranje mrežnega direktorija v primeru dodajanja novih mej (oziroma brisanja že obstoječih) precej zamudno. V naši implementaciji rešujemo ta problem podobno kot Hinrichs [Hin85a, Hin85b] z uvedbo dvonivojskega mrežnega direktorija. Zgornji nivo (v nadaljnjem besedilu glavni direktorij) skupaj s pripadajočimi skalami predstavlja skalirano verzijo prvotnega enonivojskega mrežnega direktorija in je dovolj majhen, da ga med obdelavo hranimo v

glavnem pomnilniku. Spodnji nivo, ki ustreza prvotnemu direktoriju, je razdeljen na strani in se nahaja na disku. Vsaka stran vsebuje del prvotnega direktorija (v nadaljnjem besedilu poddirektorij) in pripadajoče skale (v nadaljnjem besedilu subskale).

Ta organizacija omogoča, da se prestrukturiranje mrežnega direktorija omeji samo na tisto stran poddirektorija, v kateri je prišlo do spremembe, ostale strani pa niso prizadete. Obenem pa nam ta organizacija zagotavlja, da sta za dostop do iskanega zapisa (ob predpostavki, da poznamo vrednosti atributov, po katerih je zgrajena mreža), potrebna dva dosega na disk: branje ustrezne strani poddirektorija in branje ustreznega podatkovnega bloka.

Dvonivojska organizacija mrežnega direktorija zahteva dopolnitev prej opisanih postopkov za razcep oziroma zlivanje podatkovnih blokov, saj se lahko spremembe zaradi dodajanja oziroma odvzemanja mej odražajo na dveh nivojih: v glavnem direktoriju in v poddirektoriju. Pri razcepu podatkovnega bloka je treba najprej ažurirati subskale in poddirektorij na tekoči strani, to pa lahko privede do prekoračitve velikosti strani. V tem primeru moramo razcepiti stran ter ažurirati glavni direktorij in skale glavnega direktorija.

Podobno velja tudi pri zlivanju podatkovnih blokov: Eliminacija nepotrebne meje v poddirektoriju povzroči zmanjšanje zasedenosti tekoče strani, kar ima lahko za posledico potrebo po zlitju dveh strani, to pa se spet odraža v prestrukturiranju glavnega direktorija in njegovih skal.

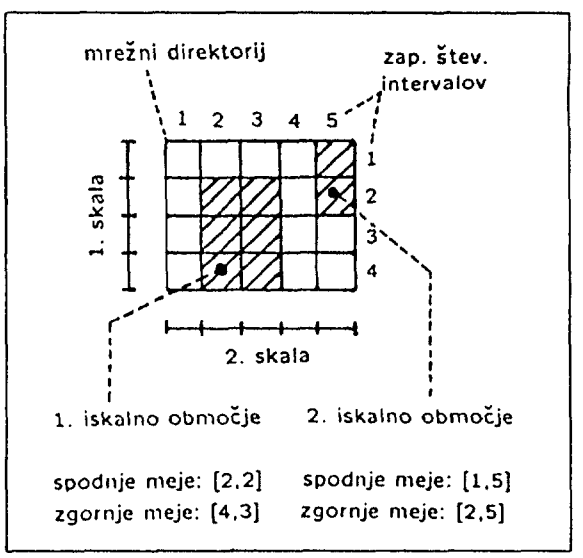
3. POSTOPEK OMEJEVANJA ISKALNEGA OBMOČJA

Učinkovita realizacija poizvedovanj zahteva, da se pri iskanju podatkov omejimo samo na tisti del podatkovnega prostora, v katerem se nahajajo zapisi, ki ustrezajo zahtevanim pogojem. Tipičen primer poizvedovanja, pri katerem pridejo lastnosti mrežnih datotek še posebej do veljave, je intervalsko poizvedovanje. Pri tem poizvedovanju iščemo tiste zapise z datoteke d, ki imajo vrednosti znotraj (za vsak atribut posebej) predpisanega intervala. V mrežnih datotekah dobi to poizvedovanje enostavno geometrično interpretacijo: vsi zapisi, ki usterzajo zahtevanim pogojem, ležijo v podatkovnem prostoru znotraj hiperparalelepipeda, katerega stranice so določene z vrednostmi intervalov v posameznih dimenzijah.

Operacija relacijske algebre .selekcija (v nadaljnjem besedilu jo označujemo v skladu z notacijo iz [Ull80] kot σ_F) predstavlja posplošitev enostavnega intervalskega poizvedovanja. Kot rezultat omejevanja iskalnega območja dobimo pri tej operaciji seznam iskalnih območij, ki v splošnem lahko obsega večje število hiperparalelepipedov.

Formula F, ki ji morajo zadoščati iskani zapisi, je sestavljena iz:

- predikatov oblike *atr op konst*, kjer je *atr* ime atributa, *op* aritmetični operator (=,<>,<=,>=,< ali >) in *konst* konstanta, katere vrednost pripada isti domeni kot vrednost atributa *atr*;
- logičnih operatorjev *and*, *or* in *not*, s katerimi so posamezni predikati povezani med seboj.



Slika 1: Predstavitev seznama iskalnih območij v mrežni datoteki z dvema atributoma

V naši implementaciji operacije selekcija je formula F interno predstavljena v postfiksni obliki, za določitev iskalnega območja pa uporabljamo sklad seznamov iskalnih območij. Vsako iskalno območje v seznamu ima obliko hiperparalelepipeda in je predstavljeno z dvema poljema: poljem spodnjih mej iskalnega območja in poljem zgornjih mej iskalnega območja. Polje spodnjih mej vsebuje zaporedne

številke najnižjih intervalov, polje zgornjih mej pa zaporedne številke najvišjih intervalov v posameznih dimenzijah (glej sliko 1).

Omejevanje podatkovnega prostora, ki ga je treba preiskati, da bi našli zapise, ki ustrezajo formuli F, zahteva dve vrsti aktivnosti: obdelavo predikatov in obdelavo logičnih operatorjev.

Obdelava predikata

Pri obdelavi predikata določimo iskalno območje, ki ustreza tekočemu predikatu, in postavimo to iskalno območje na vrh sklada. Vsak predikat, razen tistega, pri katerem je operator *op* neenačaj, prispeva k omejitvi iskalnega območja v tisti dimenziji, ki ji pripada atribut *atr.* Iskalno območje za posamezen predikat določimo v skladu z naslednjim algoritmom:

inicializiraj iskalno območje predikata tako, da obsega celoten direktorij;
s pomočjo ustrezne skale poišči interval, ki pripada konstanti *konst*;
case operator *op* of
- >: spodnja meja iskalnega območja je enaka zaporedni številki intervala (ali za 1 večja, če konstanta sovpada z zgornjo mejo dobljenega intervala);
- >=: spodnja meja iskalnega območja je enaka zaporedni številki intervala;
- <,<=: zgornja meja iskalnega območja je enaka zaporedni številki intervala;
- =: spodnja in zgornja meja iskalnega območja sta enaki zaporedni številki intervala

end;

Isti algoritem se uporablja tako na nivoju glavnega direktorija kot na nivoju poddirektorija, s to razliko, da na nivoju poddirektorija pred iskanjem intervala preverimo, če vrednost konstante sploh pade v tisto območje, ki ga pokrivajo subskale.

Obdelava logičnega operatorja

Pri obdelavi logičnega operatorja se v primeru operatorja *and* izvrši presek seznamov iskalnih območij, v primeru operatorja *or* pa unija seznamov iskalnih območij, ki se trenutno nahajata na vrhu sklada.

Pri uniji iskalnih območij lahko dobimo seznam iskalnih območij, ki se med seboj prekrivajo. Ta problem rešujemo tako, da med postopkom pregledovanja mrežne datoteke vzdržujemo dva bitna vektorja: vektor pregledanih strani in vektor pregledanih podatkovnih blokov. S tem dosežemo, da se neka stran oziroma podatkovni blok, ki nastopa v dveh ali več različnih hiperparalelepipedih, pregleda samo enkrat.

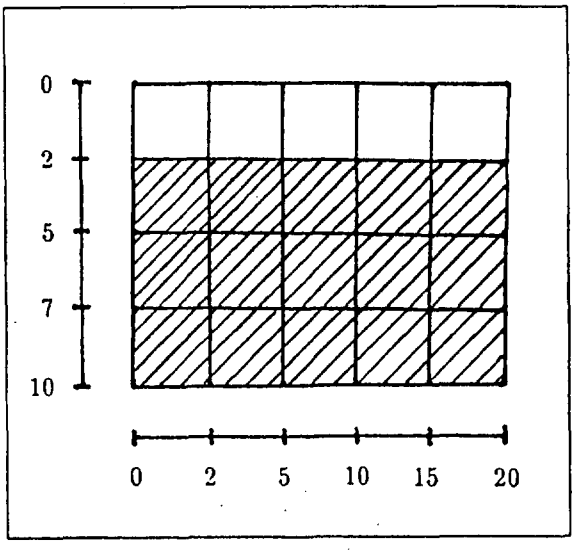
Naša implementacija zaenkrat ne podpira operatorja *not*, saj se je temu operatorju moč izogniti z ustreznim preoblikovanjem formule F. Lahko pa bi operator *not* implementirali tako, da bi poiskali negacijo seznama iskalnih območij, ki se trenutno nahaja na vrhu sklada. Negacija mora poleg komplementov prvotnih iskalnih območij vključevati tudi mejne intervale prejšnjih iskalnih območij.

Postopek omejevanja iskalnega območja v mrežni datoteki z dvema atributoma a_1 in a_2 je za formulo F

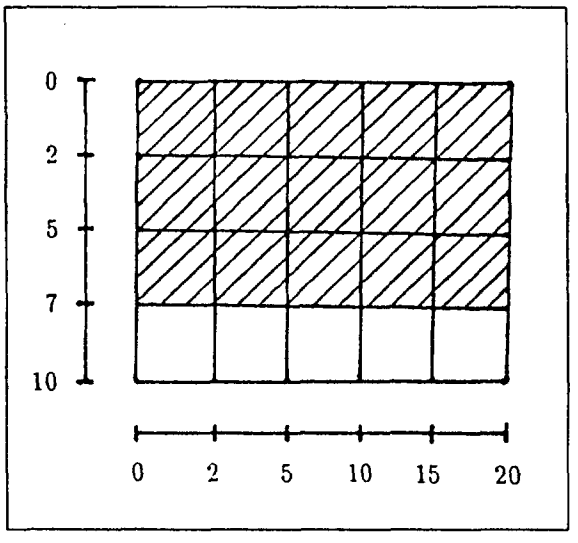
$(a_1>3)$ *and* $(a_1<6)$ *and* $((a_2=4)$ *or* $(a_2>14))$

prikazan na slikah 2a do 2g.

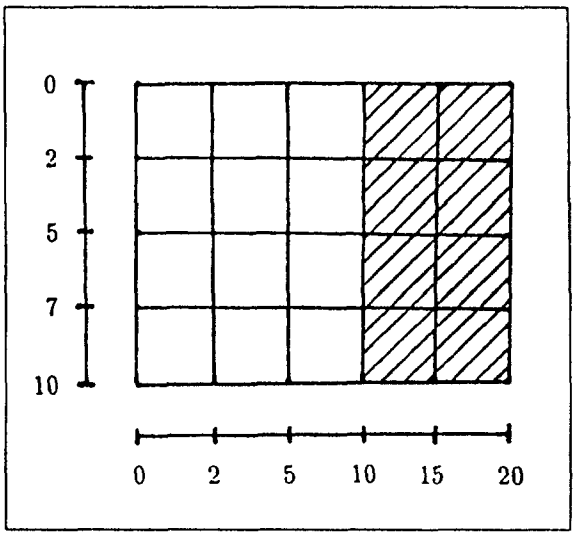
Na sliki 2g je poleg iskalnega območja vrisano tudi poizvedovalno območje, t.j. tisti del podatkovnega prostora, v katerem se dejansko nahajajo zapisi, ki zadoščajo formuli F. V splošnem je poizvedovalno območje podmnožica iskalnega, kar pomeni, da pri ovrednotenju poizvedovanja največkrat pregledamo nekoliko večji del podatkovnega prostora, kot bi bilo potrebno. Zato moramo imeti na voljo mehanizem za izločanje tistih zapisov, ki sicer ležijo v iskalnem območju, niso pa znotraj poizvedovalnega območja.



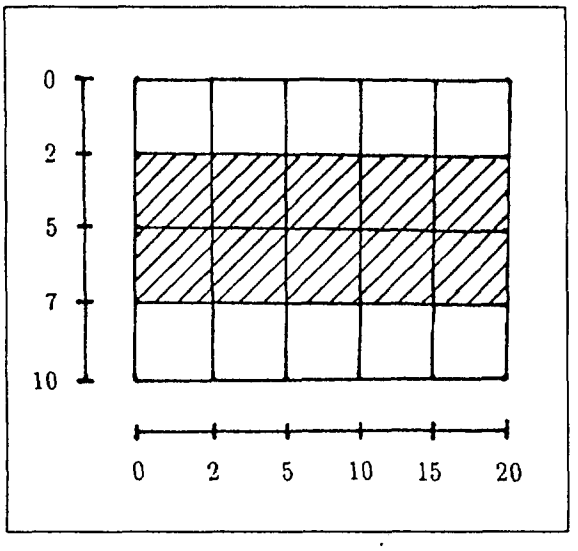
Slika 2a: Predikat $a_1>3$



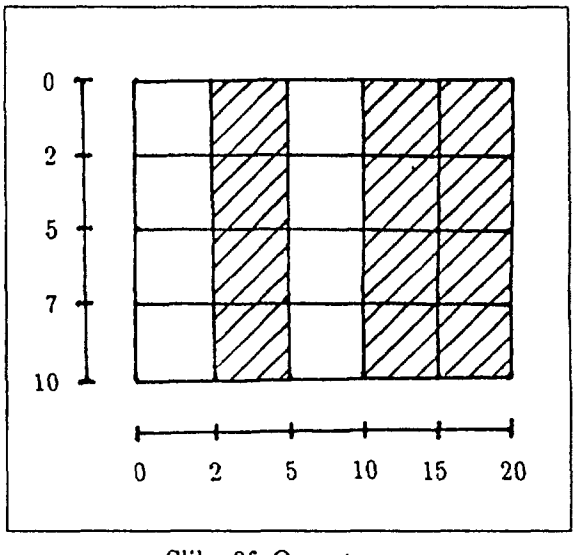
Slika 2b: Predikat $a_1<6$



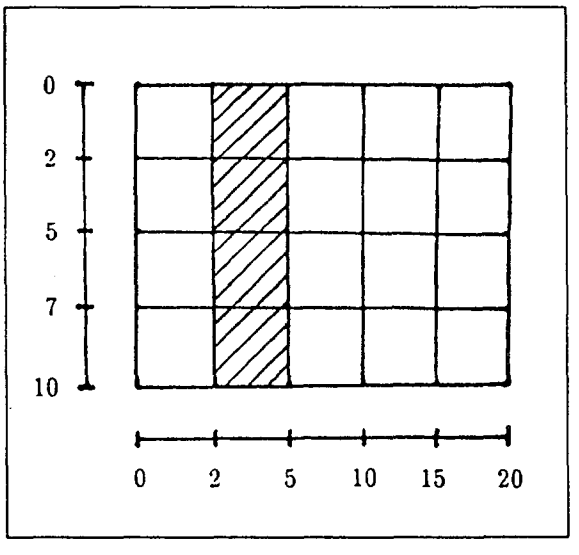
Slika 2e: Predikat $a_2>14$



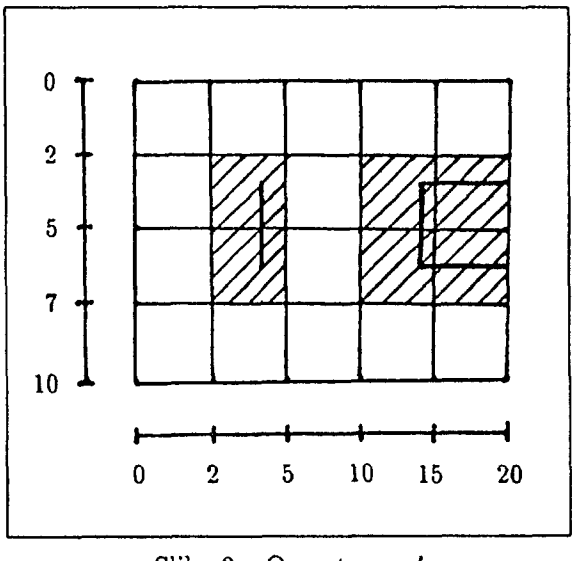
Slika 2c: Operator *and*



Slika 2f: Operator *or*



Slika 2d: Predikat $a_2=4$



Slika 2g: Operator *and*

4. ALGORITEM ZA IMPLEMENTACIJO SELEKCIJE

Implementacija selekcije zahteva, da se zgoraj opisani postopek omejevanja iskalnega območja uporabi na dveh nivojih: na nivoju glavnega direktorija in (za vsako stran posebej) na nivoju poddirektorija. Ko je obdelava formule F končana, dobimo vsakokrat na vrhu sklada ustrezen seznam iskalnih območij. Strani, ki se nahajajo v iskalnem območju glavnega direktorija, in podatkovne bloke, ki se nahajajo v iskalnem območju poddirektorija, lahko obdelujemo v kakršnemkoli vrstnem redu, saj v mrežnih datotekah - v nasprotju z običajnimi načini organizacije podatkov (glej n.pr. [BE77, SACLP79, Yao79]) - problem izbora najcenejše pristopne poti ne obstaja.

V našem programskem paketu je selekcija realizirana v skladu z naslednjim algoritmom:

```
na podlagi formule F določi seznam iskalnih
območij glavnega direktorija;
while niso obdelana vsa iskalna območja gl.dir. do
 while niso obdelane vse strani iz tekočega
  iskalnega območja do
   if stran je v celoti v poizvedovalnem območju
   then
    obdelaj vse podatkovne bloke, ki pripadajo
    tej strani {vsi zapisi v teh podatkovnih
    blokih zadoščajo formuli F}
   else
    begin
     na podlagi formule F določi seznam
     iskalnih območij poddirektorija;
     while niso obdelana vsa iskalna območja
     poddirektorija do
      while niso obdelani vsi podatk. bloki iz
      tekočega iskalnega območja do
       if podatk. blok je v celoti v poizvedoval-
       nem območju then
        vsi zapisi v podatkovnem bloku zado-
        ščajo formuli F
       else
        za vsak zapis posebej preveri, ali leži
        znotraj poizvedovalnega območja
   end;
```

Iz opisa algoritma je razvidno, da se testiranje pripadnosti poizvedovalnemu območju izvaja na treh nivojih: na nivoju strani, na nivoju podatkovnega bloka in na nivoju zapisa.

Če je stran v celoti v poizvedovalnem območju, so tudi vsi pripadajoči podatkovni bloki in v njih shranjeni zapisi v celoti v poizvedovalnem območju, t.j. zadoščajo formuli F. Če pa stran ni v celoti v poizvedovalnem območju, je treba omejiti iskalno območje tudi na nivoju poddirektorija in pregledati samo tiste podatkovne bloke, ki se nahajajo znotraj dobljenega seznama iskalnih območij. Vsak od teh podatkovnih blokov je lahko v celoti ali pa le delno v poizvedovalnem območju. V zadnjem primeru je treba za vsak zapis posebej preveriti, ali zadošča formuli F.

Preverjanje pripadnosti poizvedovalnemu območju izvajamo (podobno kot omejevanje iskalnega območja) s pomočjo sklada, le da so sedaj elementi sklada logične spremenljivke. Vsak predikat formule F povzroči, da se na vrh sklada postavi vrednost *true* ali *false*, medtem ko operatorja *and* in *or* izvršita konjunkcijo oziroma disjunkcijo vrednosti, ki se trenutno nahajata na vrhu sklada, in dobljen rezultat zopet postavita na sklad. Ko je obdelava formule F končana, nam vrednost na vrhu sklada pove, ali je tekoča stran (podatkovni blok, zapis) v celoti znotraj s to formulo določenega poizvedovalnega območja.

Pri ugotavljanju, ali je tekoča stran v celoti v poizvedovalnem območju, privzamemo, da je vrednost atributa *atr*, ki nastopa v predikatu formule F, enaka vrednosti spodnje oziroma zgornje meje območja strani v tisti dimenziji, ki ji pripada atribut *atr*. Če je operator *op* > ali >=, uporabimo vrednost spodnje meje, če pa je operator *op* < ali <=, uporabimo vrednost zgornje meje območja strani. Kadar je operator *op* neenačaj, morata biti vrednosti obeh mej območja strani (spodnje in zgornje) manjši ali večji od konstante *konst*. Če pa je operator *op* enačaj, vedno velja, da stran ni v celoti v poizvedovalnem območju.

Analogno postopamo pri obdelavi podatkovnih blokov, le da sedaj namesto vrednosti spodnje in zgornje meje območja strani upoštevamo vrednosti spodnje in zgornje meje območja podatkovnega bloka.

Pri ugotavljanju, ali je zapis znotraj poizvedovalnega območja, pa se *atr* nadomesti z vrednostjo tega atributa v tekočem zapisu.

5. ALGORITMI ZA IMPLEMENTACIJO SPLOŠNIH POIZVEDOVANJ

Splošna poizvedovanja v relacijskih podatkovnih bazah so najpogosteje sestavljena iz operacij selekcije, projekcije in stika [BE77].

Za učinkovito realizacijo stika je pomembno, da velikost relacij pred stikanjem čim bolj omejimo. Zato pri implementaciji splošnih poizvedovanj (če se le da) uporabljamo hevristično pravilo "Izvrši selekcijo takoj, ko je mogoče" [SC75, WY76, Ull80, KS86].

V običajni fizični organizaciji podatkov je možnost za izvršitev selekcije σ_F pred stikom odvisna od tega, ali obstajajo indeksi po atributih, ki nastopajo v formuli F. Vsak od teh indeksov omogoča omejitev iskalnega območja po enem samem atributu, kar pomeni, da moramo za vse zapise iz tako dobljenega iskalnega območja dodatno preverjati, ali ustrezajo preostalim predikatom iz formule F. Če imamo na voljo večje število indeksov, pregledujemo relacijo po tistem indeksu (pristopni poti), ki terja najmanj stroškov v smislu števila dosegov na disk in obsega procesiranja v glavnem pomnilniku [SACLP79]. Lahko pa s preseki razpoložljivih indeksov zreduciramo iskalno območje [AD75], vendar na račun dodatnega procesiranja, ki je potrebno za izvajanje presekov.

Sistematičen pregled različnih strategij za evaluacijo splošnih poizvedovanj z uporabo običajne fizične organizacije podatkov je podan v [Yao79].

Mrežne datoteke nudijo v primerjavi z običajno indeksno-sekvenčno organizacijo podatkov naslednje prednosti:

- Vedno je možno izvršiti selekcijo pred stikom. Omejitev iskalnega območja lahko izvršimo ne samo po enem, temveč po vseh atributih, ki nastopajo v formuli F (ob predpostavki, da je mreža generirana po vseh atributih relacije).

- Odpade problem pristopnih poti: V mrežnih datotekah se vsi atributi obravnavajo enakovredno, zato je vseeno, v kakšnem vrstnem redu pregledujemo strani in podatkovne bloke.

V okviru naše implementacije mrežnih datotek smo razvili dva algoritma, ki omogočata evaluacijo splošnih poizvedovanj naslednje oblike:

$$\pi(\sigma_{F_1}(R) \underset{i\theta j}{|\times|} \sigma_{F_2}(S))$$

Posamezni simboli imajo (v skladu z [Ull80]) naslednji pomen:

- R in S sta imeni relacij, ki nastopata v poizvedovanju.
- σ_{F_1} predstavlja selekcijo nad relacijo R, σ_{F_2} pa selekcijo nad relacijo S.
- π pomeni projekcijo.
- $|\times|$ predstavlja θ-stik, kjer je θ aritmetični primerjalni operator. Če je θ enačaj, imamo opravka s posebno vrsto stika, ki mu rečemo stik na osnovi enakosti (angl. equijoin).

Prvi algoritem temelji na bločno orientiranem gnezdenju iteracij, drugi pa na stikanju parov rezin, ki vsebujejo iste vrednosti skupnega atributa.

Stik z gnezdenjem

Pri tem algoritmu najprej uporabimo formuli F_1 in F_2 ter z njuno pomočjo omejimo iskalni območji glavnega direktorija v relacijah R oziroma S. Nato obdelujemo strani iz seznama iskalnih območij relacije R in za vsako stran določimo, katere podatkovne bloke moramo pregledati.

Če je stran v celoti vsebovana v poizvedovalnem območju, pregledamo vse pripadajoče podatkovne bloke, v nasprotnem primeru pa s pomočjo formule F_1 omejimo iskalno območje poddirektorija in pregledamo samo tiste podatkovne bloke, ki pripadajo seznamu iskalnih območij.

Za vsak podatkovni blok B_R relacije R skušamo čim bolj omejiti iskalno območje relacije S. Poleg formule F_2 pri tem uporabljamo še vrednosti spodnje in/ali zgornje meje območja podatkovnega bloka B_R v tisti dimenziji, ki ustreza atributu stika. Če je operator θ enačaj, upoštevamo obe meji. V nasprotnih primerih pa uporabimo samo spodnjo mejo (ko je θ > ali >=) oziroma zgornjo mejo (ko je θ < ali <=).

Omejevanje iskalnega območja relacije S zopet izvršimo na dveh nivojih: na nivoju glavnega direktorija in na nivoju poddirektorija. Za vsak podatkovni blok B_R pregledamo samo tiste strani in podatkovne bloke relacije S, ki pripadajo tako omejenemu iskalnemu območju.

Za ta algoritem je značilno, da vsako stran in vsak podatkovni blok prve relacije pregledamo samo enkrat, medtem ko strani in podatkovne bloke druge relacije obdelujemo večkrat. Zato lahko v primeru, ko je število pomnilniških vmesnikov, ki jih imamo na razpolago za hranjenje podatkovnih blokov oziroma strani,

majhno, pride do znatnega povečanja števila dosegov na disk.

Stik po rezinah

V praksi je stik na podlagi enakosti najbolj pogost primer operacije stikanja dveh relacij. Za realizacijo te vrste stika je bila v [JSBS83] uporabljena strategija stikanja z zlivanjem, ki zagotavlja, da se vsaka stran in vsak podatkovni blok obeh relacij obdela samo enkrat. Pač pa ta strategija zahteva, da obe relaciji pred zlivanjem sortiramo, saj v mrežnih datotekah ni moč vzdrževati urejenosti relacije po določenem atributu oziroma skupini atributov.

Algoritem za stik po rezinah, ki smo ga implementirali v okviru našega programskega paketa, ne zahteva predhodnega sortiranja obeh relacij, obenem pa zagotavlja (ob predpostavki, da imamo v glavnem pomnilniku dovolj prostora, da hranimo tekoči rezini obeh relacij), da vsak podatkovni blok in vsako stran obeh relacij preberemo z diska samo enkrat.

Zasnova algoritma temelji na dejstvu, da je moč vsako mrežno datoteko razdeliti na rezine. Na nivoju glavnega direktorija so rezine določene s tisto skalo glavnega direktorija, ki pripada atributu stika. Na enak način lahko vsako rezino glavnega direktorija "razrežemo" še naprej z uporabo subskal na nivoju poddirektorija. Vsaka rezina je omejena z dvema zaporednima mejama v skali. To pomeni, da vsaka rezina vsebuje samo tiste zapise, pri katerih je vrednost atributa stika znotraj intervala, določenega s spodnjo in zgornjo mejo rezine.

Da bi izračunali stik dveh relacij, obdelujemo rezine po parih: eno rezino relacije R in eno rezino relacije S. Na ta način se računanje stika prevede na zaporedje stikov parov rezin, ki so mnogo manjše od prvotnih relacij in jih lahko v večini primerov hranimo v glavnem pomnilniku. Obdelati je treba samo tiste pare rezin, ki vsebujejo zapise z istimi vrednostmi atributa stika.

Par rezin lahko staknemo na različne načine (z zlivanjem, z gnezdenjem iteracij itd.). V naši implementaciji smo se odločili za uporabo binarnega drevesa kazalcev na zapise prve rezine, ki ga zgradimo v glavnem pomnilniku in s tem pospešimo iskanje zapisov, pri katerih se vrednosti atributa stika ujemajo.

Čeprav lahko isti podatkovni blok nastopa v več zaporednih rezinah, ga beremo z diska samo enkrat. To dosežemo na ta način, da vsako rezino preberemo v glavni pomnilnik v dveh korakih:

- Najprej sprostimo prostor, ki ga zasedajo podatkovni bloki, ki ne pripadajo naslednji rezini. V pomnilniku tako ostanejo samo tisti podatkovni bloki prejšnje rezine, ki pripadajo tudi naslednji. Organizacija mrežnih datotek zagotavlja, da nobenega od sproščenih podatkovnih blokov v nadaljnjem postopku ne bomo več potrebovali.

- Nato preberemo z diska še preostale podatkovne bloke, ki spadajo v tekočo rezino, a jih še ni v glavnem pomnilniku.

Vsak podatkovni blok se tako prebere z diska samo takrat, ko ga prvič potrebujemo, nato pa ostane v glavnem pomnilniku toliko časa, dokler ne obdelamo vseh rezin, v katerih se nahaja.

Podrobnejši opis obeh algoritmov je podan v [Mah89].

6. OPIS PROGRAMSKEGA PAKETA

Programski paket za vzdrževanje mrežnih datotek in izvajanje poizvedovanj na osnovi operacij relacijske algebre, ki smo ga razvili v okviru naših raziskav, je napisan v programskem jeziku Turbo Pascal [Bor87] in je prilagojen za uporabo na osebnih računalnikih tipa IBM PC XT/AT.

Vsaka mrežna datoteka je predstavljena z dvema Pascal-skima datotekama v skladu z naslednjimi deklaracijami:

```
const DolzinaBloka=512;
type Blok=array [1..DolzinaBloka] of char;
var PodMD,DirMD: file of Blok;
```

Datoteka PodMD hrani podatke, t.j. zapise, ki pripadajo neki relaciji. V vsakem bloku je lahko večje število zapisov, maksimalna dolžina zapisa pa ne sme presegati dolžine bloka.

Na datoteki DirMD je shranjen mrežni direktorij, ki je razdeljen na strani. Vsak zapis datoteke DirMD vsebuje eno stran poddirektorija. Poleg poddirektorija so na datoteki DirMD shranjeni tudi podatki o

glavnem direktoriju, podatki o strukturi zapisa (tip, začetek in dolžina vsakega atributa) ter vektorja zasedenih strani in podatkovnih blokov.

Ob kreiranju mrežne datoteke moramo definirati strukturo zapisa. Za vsak atribut navedemo njegovo ime, tip in dolžino. Če želimo po tem atributu graditi mrežo, pa moramo podati tudi minimalno in maksimalno vrednost, ki jo ta atribut lahko zavzame. Ti dve vrednosti se uporabita kot spodnja in zgornja meja v pripadajoči skali.

Naša implementacija podpira zaenkrat naslednje tipe atributov: *byte, integer, word, real* in *character (niz znakov)*. Dolžina atributov tipa *byte, integer, word* in *real* je določena skladno z interno predstavitvijo podatkov tega tipa v Turbo Pascal-u in znaša 1, 2, 2 oziroma 6 bytov. Dolžino atributov tipa *character* določa uporabnik, vendar le-ta ne sme preseči 255 znakov.

Programski paket je sestavljen iz 8 programskih enot (angl. units) z naslednjo vsebino:

- Programska enota DEKL vsebuje deklaracije konstant, tipov in spremenljivk, ki so skupne za celoten programski paket in jih uporabljajo vse ostale enote.

- Programska enota SKUPNE obsega vse skupne podprograme, ki jih lahko kličemo iz katerekoli druge enote. Med te podprograme spadajo:

  podprogrami za pretvorbo numeričnih podatkov (tipa *byte, integer, word* ali *real*) v znakovno obliko in obratno;

  podprogrami za vpisovanje podatkov v glavni direktorij, poddirektorij in pripadajoče skale;

  podprogrami za rekonstrukcijo podatkov, ki so zapisani v glavnem direktoriju, poddirektoriju in pripadajočih skalah (n.pr. število mej, vrednost in položaj posamezne meje, vrednost in položaj posameznega elementa v glavnem direktoriju oziroma poddirektoriju, število vseh elementov glavnega direktorija oziroma poddirektorija itd.;

  podprogrami za prenos podatkovnih blokov in strani z diska v glavni pomnilnik in obratno;

  podprogrami za določanje koordinat tiste celice v mreži, v katero se preslika zapis kot točka v podatkovnem prostoru;

  podprogrami za ugotavljanje območja strani in območja podatkovnega bloka

  itd.

- Programska enota VZDR vsebuje podprograme, ki so potrebni za vzdrževanje mrežnih datotek. To so podprogrami za kreiranje, odpiranje in zapiranje mrežnih datotek. Novo datoteko lahko kreira uporabnik, lahko pa jo dobimo kot rezultat nekega poizvedovanja. V tem primeru se struktura mrežne datoteke generira avtomatsko v skladu s seznamom atributov, navedenih v sklopu projekcije.

- Programska enota VST vsebuje podprogram za vstavljanje zapisov v mrežno datoteko, v okviru katerega so implementirani vsi potrebni algoritmi za dodajanje novih mej, razcep podatkovnih blokov in razcep strani. Programski paket omogoča tako interaktivno dodajanje posameznih zapisov kot tudi dodajanje zapisov z druge datoteke, ki mora biti tipa *text*, podatki na njej pa morajo biti pripravljeni v skladu s strukturo zapisa na mrežni datoteki.

- Programska enota BRIS vsebuje podprogram za brisanje zapisov iz mrežne datoteke, vključno s podprogrami za zlivanje podatkovnih blokov, izločanje nepotrebnih mej in zlivanje strani.

- Programska enota SKPOIZ obsega podprograme, ki so skupni za različna poizvedovanja. V glavnem gre za podprograme, ki omogočajo interpretacijo formule F pri selekciji, omejevanje iskalnega območja in ugotavljanje, ali je tekoča stran, podatkovni blok oziroma zapis v celoti v poizvedovalnem območju.

- Programska enota POIZ vsebuje podprograme za izvajanje različnih poizvedovanj. Na voljo so podprogrami za izvajanje selekcije in splošnih poizvedovanj, sestavljenih iz projekcije, selekcije in stika.

  Splošna poizvedovanja, ki vključujejo stik dveh datotek, so realizirana s tremi različnimi podprogrami: z bločno orientiranim gnezdenjem iteracij, s stikanjem po rezinah glavnega direktorija in s stikanjem po rezinah poddirektorija.

Pri stikanju po rezinah glavnega direktorija se rezine formirajo samo na osnovi skal glavnega direktorija, zato je ta metoda hitrejša, vendar daje večje rezine in je primerna samo za manjše datoteke. Pri stikanju po rezinah poddirektorija pa pri oblikovanju rezin upoštevamo tudi subskale ter s tem zmanjšamo velikost posameznih rezin. Zato je ta metoda primerna za večje datoteke.

Programska enota INFO vsebuje podprograme za izpis vsebine mrežne datoteke in mrežnega direktorija. Izpisujemo lahko celotno datoteko, posamezne podatkovne bloke in posamezne zapise: Od podatkov, ki so shranjeni v okviru mrežnega direktorija, lahko izpišemo strukturo datoteke (tip, začetek in dolžino posameznih atributov), glavni direktorij in posamezne strani poddirektorija.

V to enoto spadajo tudi podprogrami za izračun raznih statističnih podatkov (n.pr. število zasednih strani in podatkovnih blokov, povprečna velikost območja strani oziroma podatkovnega bloka, povprečno število elementov v poddirektoriju itd.).

7. SKLEP

V članku so opisane osnovne značilnosti mrežnih datotek in algoritmi za implementacijo poizvedovanj na osnovi operacij relacijske algebre. Bistvena prednost mrežnih datotek je, da omogočajo učinkovito omejevanje podatkovnega prostora, ki ga je treba preiskati, da bi našli podatke, ki ustrezajo posameznim poizvedovanjem. Postopek omejevanja iskalnega območja predstavlja izhodišče za realizacijo selekcije.

Splošna poizvedovanja, ki zahtevajo stikanje dveh datotek, lahko realiziramo na osnovi različnih strategij. V članku sta opisana dva algoritma: prvi temelji na bločno orientiranem gnezdenju iteracij, drugi pa na stikanju parov rezin. S pomočjo drugega algoritma se postopek stikanja dveh obsežnih relacij pretvori v zaporedje stikov mnogo manjših relacij. S tem se izognemo problemu sortiranja relacij, ki je potrebno pri stiku z zlivanjem. Poleg tega pa dosežemo, da se vsaka stran in vsak podatkovni blok obeh relacij prebere z diska samo enkrat ob predpostavki, da je v glavnem pomnilniku dovolj prostora za en par rezin.

Vse opisane algoritme smo implementirali v okviru programskega paketa, ki omogoča vzdrževanje mrežnih datotek in izvajanje ustreznih poizvedovanj. Način predstavitve mrežnih datotek in zgradba programskega paketa sta opisana v zadnjem razdelku.

8. SODELAVCI

Implementacijo algoritma za brisanje zapisov je v okviru svojega diplomskega dela opravila študentka Fakultete za elektrotehniko in računalništvo Nataša Žabkar. Študent Božo Urh pa v svoji diplomski nalogi proučuje vpliv različnih načinov oblikovanja mreže na velikost mrežnega direktorija in performanse posameznih poizvedovanj. Obema se za njun prispevek k obravnavani problematiki najtopleje zahvaljujem.

LITERATURA:

[AD75] M.M. Astrahan, D.D. Chamberlin: IMPLEMENTATION OF A STRUCTURED ENGLISH QUERY LANGUAGE, Communications of the ACM, Vol. 18, No. 10, October 1975

[BE77] M.W. Blasgen, K.P. Eswaran: STORAGE AND ACCESS IN RELATIONAL DATA BASES, IBM Systems Journal, No. 4, 1977

[Bor87] TURBO PASCAL OWNER'S HANDBOOK, Version 4.0, Borland International, 4585 Scotts Valley Drive, Scotts Valley, CA 95066, ISBN 0-87524-170-0, 1987

[Hin85a] K.H. Hinrichs: THE GRID FILE SYSTEM: IMPLEMENTATION AND CASE STUDIES OF APPLICATIONS, Diss. ETH Nr. 7734, Zurich, 1985

[Hin85b] K.H. Hinrichs: IMPLEMENTATION OF THE GRID FILE: DESIGN CONCEPTS AND EXPERIENCE, BIT 25, 1985

[KS86] H.F. Korth, A. Silberschatz: DATA-

BASE SYSTEMS CONCEPTS, McGraw-Hill, 1986

[JSBS83] S. M. Joshi, S. Sanyal, S. Banerjee, S. Srikumar: USING GRID FILES FOR A RELATIONAL DATABASE MANAGEMENT SYSTEM, Technical Report No. 11, Speech and Digital Systems Group, Tata Institute of Fundamental Research, Bombay, 1983

[Mah89] V. Mahnič: COMPUTING JOINS OF RELATIONS USING GRID FILES AS A PHYSICAL STORAGE STRUCTURE, Zbornik del 11. mednarodnega simpozija "Kompjuter na sveučilištu", Cavtat, junij 1989

[NHS84] J. Nievergelt, H. Hinterberger, K.C. Sevcik: GRID FILE: AN ADAPTABLE, SYMMETRIC MULTIKEY FILE STRUCTURE, ACM Transactions on Database Systems, Vol. 9, No. 1, March 1984

[SACLP79] P.G. Selinger, M.M. Astrahan, D.D. Chamberlin, R.A. Lorie, T.G. Price: ACCESS PATH SELECTION IN A RELATIONAL DATABASE MANAGEMENT SYSTEM, Proceedings of the ACM SIGMOD International Conference on the Management of Data, 1979

[SC75] J.M. Smith, P.Y. Chang: OPTIMIZING THE PERFORMANCE OF A RELATIONAL ALGEBRA DATABASE INTERFACE, Communications of the ACM, Vol. 18, No. 10, October 1975

[Sch86] E. Schikuta: THE GRIDFILE: A DATA STRUCTURE FOR RELATIONAL DATABASE SYSTEMS, Zbornik del 8. mednarodnega simpozija "Kompjuter na sveučilištu", Cavtat, maj 1986

[Ull80] J.D. Ullman: PRINCIPLES OF DATABASE SYSTEMS, Computer Science Press, Potomac, Maryland, 1980

[WY76] E. Wong, K. Youssefi: DECOMPOSITION - A STRATEGY FOR QUERY PROCESSING, ACM Transactions on Database Systems, Vol. 1, No. 3, September 1976

[Yao79] S.B. Yao: OPTIMIZATION OF QUERY EVALUATION ALGORITHMS, ACM Transactions on Database Systems, Vol 4., No. 2, June 1979

NEURAL NETWORK BASED PARALLEL EXPERT SYSTEM

INFORMATICA 3/89

Keywords: neural network, parallel processing, expert system

Saša Prešeren, Ludvik Gyergyek,
Anton P. Železnikar, and Sonja Jerman,
Iskra Delta and
Jožef Stefan Institute Ljubljana

ABSTRACT This paper describes a proposal for an expert system which is based on a neural network principles. Properties of a neural network based expert system are pointed out, the structure of neural network is formulated, processing elements are proposed and the formulation of an expert system with the neural network properties is suggested. The model is supposed to use a parallel computer with distributed processing capability, distributed memory and distributed knowledge. The "forgetting" and the "learning" functions are proposed for adaptability of the neural network based parallel expert system.

PARALELNI EKSPERTNI SISTEM, KI TEMELJI NA NEVRALNI MRE2I Članek podaja predlog ekspertnega sistema, ki temelji na principih nevralne mreže. Poudarjene so značilnosti ekspertnega sistema. ki temelji na nevralni mreži, formulirana je struktura nevralne mreže, predlagani procesni elementi in sugerirana formulacija ekspertnega sistema, ki ima lastnosti nevralne mreže. Model naj bi uporabljal paralelni računalnik s porazdeljenimi procesnimi sposobnostmi, porazdeljenim pomnilnikom ter porazdeljenim znanjem. Predlagani sta funkciji učenja in pozabljanja, ki omogočata prilagodljivost paralelnega ekspertnega sistema.

1. INTRODUCTION

Nowadays we are faced with a great interest in a neural network modeling. Neurocomputing is considered as a nonalgorithmic approach to information processing (2). This surprising fact of nonalgorithmic approach is possible because all procedural capabilities of an artificial neural network are incorporated and hidden in a special purpose hardware. Such special purpose hardware for implementing neural network algorithms is already available for character recognition (1), for analog VLSI networks for vision systems (3) and others. The learning laws, the scheduling functions and the transfer functions are in those systems realized in hardware and incorporate equations which modify the adaptive coefficients, called weights in a processing element's local memory. In this way the neural network gains adaptability and has a so called self organizing capability.

The most interesting property of neural networks is the interconnection of a large number of processing elements, even if these elements are represented by a simplified function. Different approaches for a network representation have been considered as for example a multilevel neural network with selforganizing capability and forward/backward signal flow routing as well as distributed network of individual processing elements with good message passing capabilities (5).

In this paper we are not interested in biological details of living neural structures. Rather we use the idea of a neural network structure, organization and functioning and implement a computer system with similar properties but using powerful processors. Very important is to identify those properties which make a multiprocessor based expert system to act like a neural network.

2. PROPERTIES OF NEURAL NETWORK BASED EXPERT SYSTEM

Most research work in neuralcomputing is focused to the implementation of a fine grain special purpose hardware which is performing some intelligent subfunction. Processing elements have to be "adaptive" as well as capable of communication with the neighbors. In order to incorporate the neural network properties into an expert system we first have to focus our attention to the following properties of a neural network:

- Patterns (representational features) are memorized over the whole system in a sense that properties of an object are stored in different memory modules. Therefore we say that we have a distributed information storing.

- Recognition (for example object recognition) is performed through feature extraction, that means symbolic representation rather than by memorizing the whole pattern (image).

- A neural network is fault tolerant so that if a part of a system is damaged or fails the system still works, although the probability for wrong identification is higher and/or time required for identification is longer.

- Recognition and learning are performed by local adaptation in the sense that a set of knowledge atoms is updated, a set of representational features is updated and a set

of distribution probabilities is locally updated. A new strategy is selected in real time.

- The dynamic of the knowledge network is parallel and asynchronous.

- The knowledge network contains elements which have high percentage of fuzziness. The neural network system can function with incomplete or inaccurate input data.

Many neural network based systems are problem oriented and therefore limited to a specific task. A classical neurocomputing incorporates algorithmic properties in a special purpose hardware, therefore it is limited to some problems as for example pattern recognition, handwritten character recognition, etc. Our goal is to apply neural network properties in a complex knowledge base system which may be from any expert-system domain. The rules used in solution would be generated by training data and may stay hidden to the system programmer.

A natural property of most expert systems is parallelism. The expert system is composed of a number of modules which are executed on different processors. Calling of a module depends on a data pattern. Knowledge network, together with all accompanying steps of symbolic representation, normalization and optimization gives the framework for parallelism.

3. OPERATORS AND OPERANDS

Neurocomputing can not copy the human brain structure, but much more interesting - it successfully applies some concepts of human brain. Let us formulate the elements and the structure of a neural network.

In a neural network we have:

- operators (network elements) and
- operands (stimulus, output).

A neural network is a composition of operators. Operators are influenced by a learning process, aging process, memorizing process etc. In a neural network system a processor and a knowledge system can not be separated, because a processor itself is changed or influenced, and acquires knowledge by processing information. A processing system and a knowledge system are nonseparable homogeneous units.

An operation which is performed by an operator remains unchanged by time. Therefore, a type of each operator is fixed but its power varies in intensity. The variability of a power intensity is a consequence of constant and repeating interactions of the network with the environment. A self organization, a so called learning without a teacher, is explained by varying the power intensity of a particular operator. A weight of communication determines the power of operator (intensity of connection) and depends on a design of the network and on information that a network system has learned.

Information which enters a neural network is actually processed by a set of operators. Self organization of the neural network means that after some time the same information is processed by modified operators. Operator modification is a consequence of self learning process.

In a neural based parallel expert system operators are individual processors performing a sequence of feature extractions. By time the same processors would perform a different sequence or even a different set of feature extractions, because the system has "learned" how to do the work better. Operands in the expert system are symbolic descriptions of objects which are entering the expert system.

4. PROCESSING ELEMENTS

Human brain consists of interconnected neurons which are simple processing units. Each neuron is connected to as many as 10.000 other neurons forming therefore a highly parallel processing system. Connections are sometimes even bidirectional (input and feedback) and differ in power (strong or weak). No computer can imitate such complexity of the human brain.

An artificial neuron is a classical processing element in neurocomputing. The key to success of neurocomputing is parallel processing which opened new perspectives, approaches, reasoning and problem formulation in information processing. Our approach suggests much more powerful processing elements, equivalent to several thousand neuron-like processing elements. Therefore our system does not need a great number of processing elements, since they are more powerful. Obviously the approach with powerful processing elements has lower degree of parallelism than the special purpose neuron-like network.

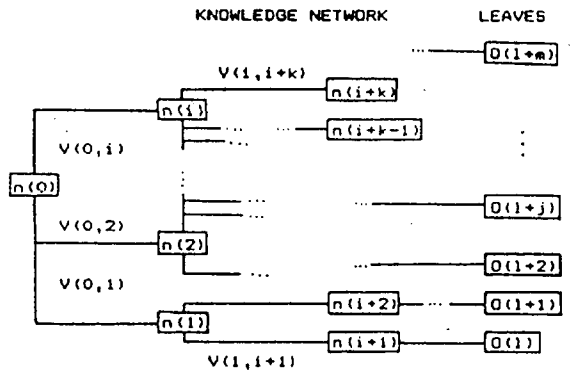
The hardware for proposed neural network based parallel expert system is a highly capable set of interconnected processors and memory modules. The array of transputers is available technology which enables a configuration of a distributed parallel computer system with neural network properties and enables learning using a sample set of learning situations.

The T800 Transputer is proposed as a processing element which incorporate a processor and a local memory. It has 4 links to the neighboring elements. A local memory is used for storing a value of adaptive coefficient (operator), important for neural network learning and fast processing. The introduction of layer-groups of processing elements which all have the same transfer function is usually not necessary because of the powerful processor. If the processing speed should be increased, a multilayer network of transputers would have to be used. The hardware frame would stay unchanged, only the software distribution to different processors would have to be modified.

5. EXPERT SYSTEM

The task of the expert system is to extract stored information or knowledge. The representation of knowledge in a neural network system was studied by Kohonen (4), using a distributed associative memory model.

Many expert systems have to perform a set of feature extractions in order to reach the goal which might be object recognition, medical diagnostics or others. A framework for those expert systems is a knowledge network. Each node in a knowledge network is reached by a feature value $V(i,j)$ which is obtained by a feature extraction. The leave node of the network is reached by the some set of feature values. This set of feature values represents a symbolic description of an object.



V(k) is a set of possible feature values at node k in a knowledge network
V(k,j) is the feature value which connects node k to node j
V(k) = (V(k,j)) j=1,...,m

Fig.1.: The nowledge network is organized in a tree structure with feature values as connections.

Each node of a knowledge network consists of a set of knowledge atoms. The initial node n(0) contains all knowledge atoms which correspond to completely unidentified object. All leave nodes consist of only one knowledge atom O(i) which corresponds to a completely identified object.

Nodes are processed in different processors. In a large knowledge network it is very likely that the number of processors is going to be smaller than the number of knowledge atoms. Therefore we perform a so called layer distribution which means that only a set of knowledge atoms branching from a father knowledge atom are stored and processed in a single processor. An example of a knowledge network from fig 1. would be processed in the following processors:

PROCESSOR	KNOWLEDGE ATOMS
1st	n(0), n(1), n(i+1),..., n(i+1+1),...
2nd	n(2), n(i+2),..., n(i+1+2),...
.	
:	
other processors	

The next step is required when the number of son knowledge atoms from a particular father knowledge atom is bigger than the available number of processors in a neural network. This is also very likely if we work with a standard parallel machine having from 20 to 500 powerful processors. Much work in parallel processing is devoted to acquire available free processors for parallel work. Different strategies have been proposed which all require some central supervisor or schedular.

6. LEARNING

The weights of interconnections are variables and have to (self)adapt to such values which determine the most probable path for fast and correct identification. Huge expert systems would be slow at the initial phase of learning because of nonoptimal search path, but would gradually became very efficient because of learning.

In a learning process we distinguish the following:

- learn to perform a job faster
- learn to enlarge a knowledge base or a problem domain.

These two types of learning require different strategies.

6.1 Strategy learning

Learning in a neural network system is performed by a set of learning examples. The resulting output is compared with the desired output. A self learning in the neural networks is achieved by using a number of methods which are recommended for different applications (5). For self learning of an expert system we are proposing another method which is based on experience of the neural network system.

Learning in order to perform a task faster requires only modification of probability coefficients c(i) in the local memories of individual processors. The coefficient c(i) is a pointer to different feature extractors in the network. At each knowledge atom only those coefficients c(i) exist which represent some object. For instance if a knowledge atom n(i) has connections to knowledge atoms n(k) and n(k+1), than only two coefficients c(k) and c(k+1) will exist in the knowledge atom n(i). Each processor performs only that feature extractor for which the coefficient c(i) has the highest value.

In the begining the probability coefficients are equivalent to c(i) = 1. After a feature V(i) is identified by a processor i, the new probability coefficient c(i) equals

```
        c(i)/2   if c(i) > 1  (forgetting func.)
        c(i)     if c(i) = 1
c(i) = c(i)      if (c(i) = c(max) and
                     V(i) was selected)
        c(i)/2+1 if V(i) was selected  (learning
                                        func.)
```

By this algorithm the changing of a strategy is performed very fast, coefficients are stored in local memory and a change of the strategy is not influenced by stohastic events. The forgetting function and the learning function are important for selfadaptability of the network.

6.2. Knowledge learning

Learning of a new knowledge or enlarging a problem domain involves much more complex procedure. This is:

- formulating the new knowledge,
- formulating the new feature values,
- establishing connections in the recognition tree,
- distributing the process to processors for parallel processing.

Several different applications as for example simulation of complex socio-economic systems, simulation of ecologic processes in natural environment, computer vision, speech recognition and some others are easier to formulate as a neural network based parallel expert system. The nature of all these applications is parallelism and complexity. Neural network approach enables easy distribution of the problem to a parallel computer system, enables step by step approach in defining and adding details of a model, gaining in that way the exactness as well as

accuracy of the model and not loosing the applicability and a speed of processing. We are considering simulation of other complex biological systems with special interest in learning, self organization, memorizing and fault tolerance.

7. CONCLUSION

Implementation of the neural network with standard microcomputer components has good price performance ratio. The capability of reformulating or upgrading the functions is much higher because neural network principles are implemented in software rather than in hardware. We have proposed a neural network based parallel expert system on transputer processors and described a problem parallelization and self learning function, based on the local coefficient adaptability.

The limitation of our proposal is problem decomposition which might be complex for some sophisticated expert systems. However, we think that a computer supported interactive decomposition have to be developed in order to faciliate knowledge network decomposition.

Having in mind the future trends in parallel processing

8. REFERENCES

(1) H. P. Graf et al., "VLSI Implementation of a Neural Network Model", Computer, IEEE Computer Society, March 1988, pp 41-49.

(2) R. Hecht-Nielsen, "Neurocomputing: Picking the Human Brain", IEEE Spectrum, March 1988, pp 36-41.

(3) J. Hutchinson et al., "Computer Motion Using Analog and Binary Resistive Networks", Computer, IEEE Computer Society, March 1988, pp 52-63.

(4) T. Kohonen, "Associative Memories and Representations of Knowledge as Internal States in Distributed Systems", European Seminar on Neural Computing, London, 1988.

(5) P. Treleaven, "Parallel architectures for neurocomputers", European Seminar on Neurocomputing, London, 1988.

forum informationis

Slovensko društvo informatika
FORUM INFORMATIONIS

Slovensko društvo informatika je priredilo dne 17.5.1989 srečanje FORUM INFORMATIONIS z naslovom:

Pogoji razvojno - raziskovalnega dela v računalništvu in informatiki v podjetjih, institutih in na univerzi

z naslednjim dnevnim redom:

1. Cilji foruma v okviru Slovenskega društva informatika
2. Poročilo o odzivu na sodelovanje
3. Razprava o pogojih razvojno - raziskovalnega dela
4. Oblikovanje sklepov
5. Razno

Po končanem forumu so udeleženci razpravljali o oblikovanju interesnih delovnih skupin. Ustanovljena je bila interesna delovna skupina z naslovom: Organiziranost raziskovalnega dela na področju fundamentalnih raziskav v katero so bili predlagani Borut Jereb, Saša Prešern, Ivo Rozman, Marjan Špegel in Jure Tasič.

V nadaljevanju povzemamo oblikovanje sklepov v dokumentu "SKLEPI FORUM INFORMATIONIS" in razpravo v dokumentu "RAZPRAVA FORUM INFORMATIONIS".

SKLEPI FORUM INFORMATIONIS
==========

Slovensko društvo informatika (SDI) je dne 17.5.1989 na srečanju FORUM INFORMATIONIS z naslovom Pogoji znanstveno raziskovalnega dela v računalništvu in informatiki obravnavalo alarmantno slabe pogoje za znanstveno raziskovalno delo v računalništvu na institutih, univerzah in v podjetjih. Ugotavljamo, da nadaljevanje takih pogojev pelje v stagnacijo, siromašenje družbe in oddaljevanje od sodobnih tokov združevanja v Evropo.

Splošno mnenje udeležencev FORUMa je bilo, da je na področju računalništva in informatike potrebno doseči:

- PROFESIONALNI odnos družbe do stroke,
- večjo KONCENTRACIJO raziskovalnega kadra,
- boljšo POVEZANOST s porabniki.

Apeliramo na Republiški komite za znanost in tehnologijo, da v sodelovanju s Slovenskim društvom Informatika imenuje ekspertno skupino, katere naloga naj bi bila izdelava ekspertize stanja in strategije računalništva in informatike. FORUM INFORMATIONIS ugotavlja:

- VLOGA RAČUNALNIŠTVA IN INFORMATIKE: Računalništvo in informatika je strateško pomembna panoga za suverenost slovenskega naroda. Naša družba se ne obnaša v skladu s pomenom, ki ga ta panoga zasluži. Predlagamo, da ekspertiza analizira pomen računalništva in informatike pri nas in njen pomen za vključitev v svetovni trend tehnološkega in informacijskega razvoja.

- INFRASTRUKTURA: Slovenija nima potrebne infrastrukture za računalništvo in informatiko. Nimamo na primer niti komiteja za računalništvo in informatiko (imamo 10 drugih komitejev), nimamo instituta (imamo 33 drugih institutov), nimamo združenja pri Gospodarski zbornici Slovenije (imamo 21 drugih združenj).

- STRATEGIJA: Slovenija nima izdelane strategije računalništva in informatike. Ekspertiza naj postavi temelje strategije, in naj obravnava:

  * strategijo vključevanja v mednarodne tokove računalništva in informatike na področju raziskav, razvoja, proizvodnje, trženja in izobraževanja,

  * ustrezno koordinacijo in napotke za uveljavitev svetovnih standardov,

  * stališče do carin in kvot na čipe, module, sisteme in na rešitve na ključ

- VREDNOTENJE KVALITETE: Apeliramo na dinamično vrednotenje kvalitete raziskovalnega dela posameznikov in skupin. Jasni, v naprej določeni kriteriji o vrednotenju kvalitete raziskovalnega dela v mednarodnih merilih naj zagotavljajo sprotno selekcijo in finansiranje kvalitete. Posebej naj bo vrednoteno raziskovalno delo in posebej razvojno delo.

- KADROVSKA POLITIKA: Ugotavljamo široko krizo na področju kadrovske politike raziskovalcev in razvojnikov v računalništvu in informatiki ter predlagamo naj ekspertiza obdela problem:

  * vračanja mladih raziskovalcev v industrijo in njihova vloga,

  * izdela predlog za učinkovit dialog in kroženje kadrov med industrijo (management in razvoj) ter raziskovalnimi institucijami v obliki rednih obojestranskih predavanj, daljših izmenjav in drugih oblik sodelovanja,

  * pripravi naj vprašalnik za anketiranje raziskovalcev in razvojnikov v industriji, na institutih in na univerzah,

  * pripravi naj predlog intenzivnejše izmenjave raziskovalnih kadrov z Evropo in ostalimi razvitimi deželami,

  * pripravi naj predlog rešitve stanovanjskega problema za tiste raziskovalne kadre (goste), ki so občasno vključeni v raziskovalne programe v Ljubljani.

- SREDSTVA ITALIJANSKE VLADE: Ekspertna skupina naj prouči, razjasni in objavi vse nejasnosti v zvezi z razpisom, vrednotenjem, izborom in končnim stanjem pri kandidaturi za sredstva Italijanske vlade po programu C.

Slovensko društvo informatika predlaga, da Republiški komite za znanost in tehnologijo pri svojem delu upošteva diskusijo iz FORUMa (priloga) ter, da formiranje ekspertne skupine, njeno delo in rezultate vskladi s Slovenskim društvom informatika.

Ljubljana, 17.5.1989

Priloga: RAZPRAVA FORUM INFORMATIONIS

Zapis (povzetek) razprave FORUM INFORMATIONIS - Pogoji raziskovalnega dela v računalništvu in informatiki v podjetjih, institutih in na univerzi dne 17.5.1989.

Prešern: Dokaj skromen odziv in udeležba na FORUMu (kljub osebnemu vabilu tistih, ki vodijo raziskovalno delo pri nas) kaže na morebitni občutek nemoči ali pa morda premajhno stopnjo profesionalizma. Na poziv k pismenim predlogom so se odzvali samo prof. Marinček, dr. Murn in tov. Pintar.

Marinček: Potrebno je posvetiti veliko pozornost zasledovanju mednarodnih standardov. Izvršni svet je osnoval komisijo Evropa 92, ki med drugim tudi obdeluje področje standardizacije. SZDL ima 23.5.89 problemsko konferenco o Evropa 92. Standardizacija ni več stvar industrije ampak je tu potrebna enotna infrastruktura.

Tasič: V računalništvu in informatiki ni definiranih globalnih ciljev. Sem proti voluntarizmu pri strokovnem delu. Definirali naj bi se raziskovalno razvojni centri. Ni tudi globalne povezave z univerzo. Mladi raziskovalci se utopijo v industriji.

Jereb: Da bi izboljšal pogoje raziskovalnega dela se marsikdo začne ukvarjati s politiko v raziskavah in potem postaneš slab politik in slab strokovnjak.

Faleskini: Ključ problema je denar. Pravzaprav gre za malo ljudi, morda 100 raziskovalcev, ki delujejo po nekaj registriranih raziskovalnih institucijah (verjetno 7) in po veliki množici (morda 100) neregistriranih. Katalog opravljenih raziskav nam kaže malo rezultatov tako v hardveru kot tudi pri programski opremi (2-3 paketi za izvoz, 200-300 domačih paketov). Interesi in kapital so popolnoma razpršeni. Koordinacija o Slovenskih raziskovalnih centrih je iluzija. V Evropi bi lahko nekaj predstavljala IJS in ID. Oba se ukvarjata z vsem. Konec leta 1988 je imela ID 240 razvojnih nalog. IJS razvija hardver, grafiko, umetno inteligenco,..., in s tem imamo finansiranje malih nalog, ki so kvazi skupaj zloženi programi. Za uspeh raziskovalnega dela je treba imeti: 1. moč = kapital (avtoriteto), 2. resurse (oprema, ljudi - opreme je dovolj saj imamo na primer CAD sistem v 8 Iskrah, vsak stane po 1 milijon dolarjev) in 3. znanje. ID je 5 krat postavljala sistem spremljanja razvojnih projektov, IJS pa še tega nima. Raziskovalci z doktoratom so v manadžerstvu ničla. Predlagam: V delovnih organizacijah bo podjetništvo izkristaliziralo in sposobnejši bodo ostali. V IJS in na univerzah naj razmislijo kdo je okolje kamor naj gredo rezultati (verjetno širše kot slovenski prostor). Če za projekt ni kapitala, ga ni mogoče spraviti na svetoni trg. Predlagam, da se ID zgleduje po vzhodnih deželah (Koreja, Indonezija). Te imajo AT računalnike ne pa VAX, mikroVAX ali paralelne sisteme. Predlagam, da naj ne bo nobenih carin. Vsak izdelek pa naj bo tudi sposoben za izvoz. Predlagam, da vse doktorje skoncentriramo okrog enega izdelka (na primer neke super EGA kartice) in da smo v tem izdelku najboljši na svetovnem trgu.

Murn: Kako naj društvo prispeva k raziskovalnemu delu. Tisti, ki se ukvarjamo z računalništvom in informatiko nismo organizirani niti na republiškem nivoju, niti v GZ, nimamo instituta. Manjka nam organiziranost. Želimo tudi pospešiti prenos v industrijo. Društvo naj bi imelo banko znanja, kapitala in idej.

Prešern: V zvezi z infrastrukturo ugotavljamo lahko konkretne podatke. V Sloveniji imamo 10 republiških komitejev, 21 različnih združenj v GZ in 33 institutov. Govorimo o informacijski družbi ampak nikomur se ni zdelo vredno postaviti infrastrukture za računalništvo in informatiko. Nimamo niti komiteja, niti združenja pri GZ, niti instituta, zato je verjetno tudi taka razpršenost kadra, znanja in rezultatov. Predno pa razmišljamo o infrastrukturi je treba strokovno ovrednotiti vlogo računalništva in informatike za razvoj družbe in suverenost naroda v prihodnosti. Drugod po svetu, kjer so postavljali strategijo razvoja družbe, so videli primarno vlogo računalništva in informatike pri prehodu v sodobno informacijsko družbo.

Špegel: Družba ima tako računalništvo in informatiko kot si jo zasluži. Od 60. leta naprej smo na tem področju delali kapitalne napake in grehe. Oblast išče odgovore za odločitve v amaterskih krogih. Pustili smo se speljati v to stanje in mnogi raziskovalci so šli v trgovske firme. Stabilnost okolja je pogoj za raziskovalno delo. Odsek za računalništvo in informatiko gostuje na IJS. Kljub temu, danes ni mogoče govoriti o predlogu o ustanovitvi instituta, ker je to širša tema. Povod za te težnje je finančni bankrot. Kvalitetne raziskovalne skupine so v nevarnosti razpada. Pri razmisleku o novih organizacijskih oblikah moramo upoštevati naslednja dejstva: majhni smo in zato mora biti znanost vključena v svetovne trende (finančno in organizacijsko), znanost mora biti integralni del obeh univerz, kadrovsko in organizacijsko se morajo povezati obe univerzi, industrija in IJS. Skupni obseg vlaganj v to področje je zelo majhen. Dodatni viri naj bi bili: mednarodni raziskovalni viri in domače DO, ki se lahko strateško obnašajo. Raziskovalno delo naj bi bilo osvobojeno finančnih problemov in naj bi rezultati bili splošne vrednote. Šele od prototipa naprej naj bi izvajali razvoj v razvojnih centrih po podjetjih.

Jereb: Društvo bi lahko pomagalo pri kadrih. Pri pogovorih med industrijo in univerzo ali IJS se diskutanti ne razumejo. Tudi fakulteta je kot kurja farma, čim več producirati, saj je od tega odvisno finansiranje. Društvo bi lahko pomagalo pri stanovanjih za mlade raziskovalce.

Kogovšek: Društvo naj bi posredovalo predavatelje za gospodarstvo in prirejalo predavanja ljudi iz gospodarstva. Ljudje niso seznanjeni s svetovnimi standardi in trendi. Društvo naj forsira formiranje jeder za kompleksne probleme na primer IS za upravljanje ali pa CIM. GZ na primer ni dobila izvajalca za IS v Sloveniji. Gradi se nova CTK in društvo naj se aktivira tudi tu. Institut je danes težko ustanoviti.

Tasič: Mnogi, ki so bili vabljeni niso prišli. V profesionalnem delu je odgovornost, tega pa tu ni. Poleg tega opažam nezaupanje med raziskovalci in aplikativci.

Faleskini: 0,6% dohodka se združuje v raziskovalno skupnost (za temeljne raziskave), približno 50 milijonov dolarjev v Iskri gre letno za razvoj (aplikativni razvoj) toda samo 280.000 dolarjev za rizičen razvoj (to je za raziskave). S kadri se ne da substituirati kapital. Slovensko zaledje ni dovolj veliko zato potrebujemo joint venture za aplikativne raziskave, kapital iz EUREKE, ESPRIT in drugih. Delajmo kot Korejci in Tajvanci. Društvo naj se organizira kot društva po svetu s članarino, ki mu bo omogočala naročanje strokovnih študij.

Veber: Že dolgo časa imam novo idejo o

informacijskem stroju. Pri nas se je ne da recenzirati ker recenzije ni mogoče plačati.

Rozman: Človeški input v raziskovalno delo je majhen in nimamo kritične mase raziskovalcev za sprejemanje in izvajanje velikih projektov. V prvi letnik pri nas se vpiše 300 študentov, diplomira jih 50, 10 od teh jih naredi magisterij, eden pa morda doktorira. Društvo naj animira za študij računalništva. Pri finansiranju računalništva preko PORS je bilo računalništvo v obrobju in ta program je žalost. Nekatere naloge sploh ne sodijo v PORS.

Marinček: Delo društva bi moralo biti neplačano in društvo bi moralo biti strokovni parlament za konfrontacijo strokovnih mnenj in javne kritike. Profesionalno delo pa mora biti plačano. Društvo naj združuje raziskovalce in aplikativce v interesne skupine in naj opozori na potrebo po koordinaciji. Čitanje standardov naj bo neplačano, ne tako kot IJS, ki je predlagal 17 FTE zato, da se bo ukvarjal s standardizacijo. Slišimo, kako se pri nas hvali umetna inteligenca iz IJS. Toda rezultati teh raziskav v industriji so minimalni. Mislim tudi, naj naši raziskovalci poberejo ideje po svetu in jih pripeljejo v industrijo. Tako so nekoč delali Japonski raziskovalci. Ustanovitev instituta pa se mi zdi težka.

Špegel: Raziskovalci ne moremo istočasno delati fundamentalne raziskave, prototipe in delovati hkrati na tržišču. Tak primer imamo na umetni inteligenci, ko naša industrija ni imela dovolj posluha za sodelovanje. Koordinacija preko RSS in PORS je bila pa popolnoma neučinkovita in niti 5% vložene energije in naporov nismo uspeli realizirati.

Rozman: Madžarska prodaja za HP, torej sodelovanje s svetovnimi firmami.

Jereb: Ali pomeni institut novo stavbo ali povezavo, koordinacijo?

Špegel: V perspektivi tudi stavbo.

Murn: Forum ima pomembno vlogo, saj združuje raziskovalce iz vseh krogov.

Špegel: Za akcije nimamo dolgo časa. Sesedajo se centri intelektualne moči. Raziskovalci, magistranti si ne morejo ustvariti življenskih pogojev, odhod v industrijo ni atraktiven in odhajajo v tujino in v privatne firme.

Kogovšek: V privatnih firmah delujejo na PC-jih. Kritična masa bo kmalu pona. Takrat se bo to ustavilo.

Rožič: Tisti ki jih ni na Forumu očitno delajo zase, ne pa za izboljšanje splošnega stanja.

Mekinda: Oblikujmo zaključke. Mislim, da smo ugotovili, da ni družbene infrastrukture za področje informatike. Opredeliti moramo tudi kaj bomo na področju informatike delali. Nadalje se moramo dogovoriti kako to realizirati, vključno z mednarodno udeležbo. Treba je tudi definirati interes industrije in širše družbe za to področje. Društvo pa lahko deluje samo preko posameznikov in posnema podobna društva v svetu. To pomeni angažirati strokovnjake, ampak ne po devalviranih cenah, ki jih naša družba vsiljuje raziskovalnemu delu. Potrebna je profesionalizacija in ta predlog bomo poslali GZ in Izvršnemu svetu.

Prešern: Pri oblikovanju zaključkov moramo izhajati iz dejstva, da je vloga računalništva in informatike v naši družbi podcenjena. Republiški komite za znanost in tehnologijo nam je v pripravah za ta Forum prisluhnil in neuradno podprl idejo o formiranju ekspertne skupine, ki naj analizira in ovrednoti stanje ter izdela predloge za ukrepe, izdela osnutek strategije. Ta strategija naj obravnava poleg že prej omenjenih problemov tudi predlog vrednotenja kvalitete raziskovalnega in razvojnega dela na področju računalništva in informatike, z v naprej postavljenimi kriteriji. Status raziskovalca (razvojnika) in njegovo finasiranje naj bi bilo povezano s temi kriteriji. Pri kadrovski politiki mislim, da bi dobili jasne pokazatelje slike z dobro anketo vseh raziskovalcev in razvojnikov iz računalništva in informatike v industriji, institutih in univerzah. Predlagam tudi, da pozovemo bodočo ekpertno skupino, da razišče vse okoliščine v zvezi z sredstvi italijanske vlade po programu C. Ekspertna skupina naj bo imenovana v soglasju s Slovenskim društvom informatika in naj pri svojem delu upošteva razpravo iz FORUMa . Rezultate ekspertize želimo obravnavati na enem od naslednjih FORUMov.

Sledilo je oblikovanje sklepov, ki so povzeti v dokumentu "SKLEPI FORUM INFORMATIONIS".

Povzel: Saša Prešern

novice in zanimivosti news

Z ekspertnim sistemom proti propadanju podjetij (Prenos znanja v računalnike)

Ed Feigenbaum, vobčepriznani oče ekspertenga sistema, profesor računalniških znanosti na stanfordski univerzi, je skupaj s svojima sodelavkama Pamelo McCorduck in Penny Nii izdal knjigo "The Rise of the Expert Company" (Vzpon ekspertnega podjetja), ki je izšla konec prejšnjega leta. V tej knjigi opisuje, kako vizionarska podjetja uporabljajo umetno inteligenco pri povečevanju produktivnosti in dobička. Ta podjetja so s takšno ekspertno informacijo lahko uresničila znantne kapitalne prihranke in dobičke.

V nekem intervjuju je Feigenbaum izjavil, da za to knjižnje delo enostavno ni bilo mogoče najti slabega primera in zato v knjigi navaja le uspešne primere. Pravi, da je sam resda največji zagnanec na področju ekspertnih sistemov, toda to, kar je izkusil z uporabo teh sistemov, enostavno presega vsa pričakovanja.

Uspehi pa niso bili doseženi brez preseganja nasprotovanj. Feigenbaum pravi, da se mora v katerem koli podjetju pri delu z ekspertnim sistemom najprej pojaviti prvak, katerega glavni lastnosti sta zagnanost in upornost v govorjenju, prepričevanju, poudarjanju, ob dodatnem delu doma in v tajnosti. Prav to pa je lahko tudi začetek uspešne kariere. Korporativna reakcija na delo prvaka je odvisna od korporativne kulture in gospodarskih okoliščin. Npr. pri Schlumbergerju, kjer je Feigenbaum konzultant, se testira resnost nekega predloga tako, da se predlogu znova in znova nasprotuje. Vendar se te vrste projekti podpirajo neuradno in na nenavadne (neortodoksne) načine.

Propadajoči ameriški proizvajalec tovornjakov Navistar se je moral v celoti prestrukturirati in oblikovati nove konkurenčne možnosti in novo profitnost. Podjetje je bilo na robu propada in je tako dozorelo za uvedbo ekspertnih sistemov v svoje poslovanje. Pristop k uvedbi ekspertnega sistema ima tudi svojo psihologijo. Vsiljevanje sistema prek tehnikov končnim uporabnikom je neuspešno. Čustvena naklonjenost uporabnikov kot lastnikov ekspertnega sistema je bistvena. Tipična reakcija zavračnaja ekspertnih sistemov pri uporabnikih je strah pred onesposobitvijo.

Feigenbaum navaja dva glavna razloga zavračanja ekspertnih sistemov. Naloga glavnih izvršnih uradnikov podjetja (po naše direktorjev in njihovih pomočnikov) je strateška vizija podjetja, proti kateri je tveganje enega milijona dolarjev za ekspertni sistem kar sprejemljiva naložba. Tudi mlajši uslužbenci na nižjih ravneh podjetja si želijo zanimivih nalog, ki jih delo z ekspertnim sistemom prinaša. Odklanjanje ekspertnih sistemov pa se pojavlja pri direktorjih srednje ravni, ki so najbolj izpostavljeni in za katere pomeni igra z milijonom dolarjev nezaslišano tveganje. Toda to nasprotovanje srednjega sloja vodstva, s katerim se sooča prvak umetne inteligence v podjetju, je zanemarljivo v primerjavi z obrambnim obzidjem, s katerim je obdana rakavost (malignost) služb vodstvene informacije. Gre za izredno občutljiv in boleč problem. Največ jeze in zagrenjenosti vpraševalcem (graditeljem) ekspertnega sistema povzročajo neprilagodljivost, konzevativnost in obstukcionizem navadnih direktorjev obdelave podatkov.

Feigenbaum anonimno navaja, kako so se npr. pojavile zahteve, da mora biti samostojni lispovski računalnik, na katerem se izvaja ekspertni sistem, združljiv z IBM SNA ali kako so se za ekspertni sistem pojavile napačne ocene, da ta povzroča zniževanje zmogljivosti glavnega računalnika. Odkod lahko izvira sovražnost te vrste? Službe upravljavske informacije so sestopile v t.i. PC revolucijo. Naslednji napad se je pojavil iz smeri sektorjev za ekspertne sisteme, ki so želeli svoje delo opraviti brez avtorizacije služb vodstvene informacije. Toda ti pojavi so bili le začasni. Medtem je IBM že močno zakoračil v plasma ekspertnih sistemov in oddelki za informacijsko tehnologijo v podjetjih so se vdali. Sedaj ko je IBM legaliziral ekspertne sisteme, so pomisleki o tveganju odpadli.

Čeprav Feigenbaum razmišlja, da industrija prepočasi vstopa v področje ekspertnih sistemov, mu ekonomist Nathan Rosenberg razlaga: Tudi televizija, ki je bila izumljena leta 1929, je dosegla svojo uveljavitev šele leta 1948. V letu 1987 je bilo po svetu v uporabi le 1500 pravih ekspertnih sistemov. Feigenbaum tako ni le oče ekspertnega sistema, temveč postaja tudi zgodovinska osebnost ekonomije. Gre za prepričanje, ki ga skupaj s Feigenbaumom imajo tudi industrijski ekonomisti in vodstveni misleci.

Feigenbaum trdi, da sta dve tretjini današnjih proizvodnih stroškov še nedotaknjeni. S tem so mišljeni na znanju osnovani stroški poslovanja z nabavo, načrtovanjem, inženiringom, zagotavljanjem kakovosti, upravljavsko koordinacijo in kontrolo, odločanjem in informacijskim pretokom, prodajo, uslugami in s finančnim vodenjem. Tu je v bistvu še prazen prostor produktivnosti, ki mu je potrebna le avtomatizacija, da bi postal dejavnik znanja. Ekspertni sistemi so ključ te avtomatizacije, ker pospešujejo vse pomembne kompetitivne komponente "pametnejšega opravljanja dela". Ekspertni sistemi tako niso le ojačevalniki znanja, temveč so prava orodja produktivnosti podjetja danes in v prihodnosti. Socialni učinki ekspertnih sistemov so primerljivi s povečanjem hitrosti avtomobilskega transporta v primerjavi s konjskim.

Feigenbauma ne zanima, kakšno taktiko bosta industrija in trgovina sprejeli, da bi omogočili tehnološko uporabo ekspertnih sistemov. Pristopi uvajanja nove tehnologije so lahko različni, npr. od večjih količin cenenih, netveganih, majhnih sistemov, razpršenih širom po podjetju (to je izkušnja Du Ponta), do

velikih, strateških sistemov (kot sta Amexov Credit Authorizer in DECov Excon za konfiguriranje računalniških sistemov). Kakor koli že, čim manjši je ekspertni sistem in čim nižje je tveganje, tem manjši so tudi prilivi iz investicije podjetja.

Feigenbaum imenuje Veliko Britanijo deželo PC-jevskih ekspertnih sistemov in pravi, da je iskanje ekspertnega sistema v Britaniji, ki bi bil realno uporaben, kot iskanje školjk pri nizki plimi. Potrebnega je obilo kopanja, da bi se morda kaj našlo. Na ameriških univerzah je tudi tehnološki transfer sijajna tradicija. Na stanfordski univerzi imajo praktično vsi profesorji svoja podjetja (izjemi sta le en germanist in en zgodovinar). Feigenbaum ima celo dve svoji podjetji za umetno inteligenco, in sicer Intelicorp in Teknowledge. Nerazvitost Britanije na področju umetne inteligence je predvsem posledica Lighthillovega poročila v letu 1971, s katerim je bil zaustavljen normalen razvoj umetne inteligence.

Tudi Japonci so opremili vrsto svojih podjetij po vsej Japonski z ekspertnimi sistemi. Medtem ko je japonska industrija osvojila lisp kot osnovni jezik, je državni raziskovalni center Icot izbral prolog. Ta poteza Icota pa je bila skrajno neprimerna, saj njegovo delo na tem področju ne bo nikoli uporabljeno. Logično programiranje (prolog) namreč ne more zmagati: preveč je nefleksibilno. Praktiki prologa želijo biti formalni in gledajo zviška na pragmatične pristope. V ideološki bitki med formalisti umetne inteligence, ki stavijo vse na predikatno logiko in onimi (fuzziji), ki tolerirajo vse, kar deluje, se pojavljajo nasprotja. Fiegenbaum se opredeljuje za druge. Njegova sled so večji in boljši sistemi, ki temeljijo na znanju. Današnji, majhni ekspertni sistemi dosegajo ničto kompentenco. Potrebno jim je neprimerno več znanja, ne pa več umske moči (zmogljivosti sklepanja). Sklepanje po analogiji je tudi naraven proces prilagajanja izkustvu.

Narediti računalnike bolj inteligentne pomeni po Feigenbaumu narediti jih občutljivejše za znanje. Obstajata dva razloga, ki govorita proti inteligenci stroja. Prvi je vitalizem, ki stoji izven okvirov današnje znanosti. Drugi od obeh razlogov je tehničen. Moč sistemov umetne inteligence izvira iz znanja, ki ga je mogoče zajameti, ne pa iz zmogljivosti strojnega sklepanja. Ekspertni sistemi morajo imeti dovolj visok potencial znanja o svetu, da bi lahko bili tudi njihovi programi dovolj prožni, prilagodljivi in inteligentni.

To kar opredeljuje umetno inteligenco je še vedno Newell-Simonova hipoteza o fizičnem simbolnem sistemu, po kateri so fizični simbolni sistemi (taki, kot možgani ali digitalni računalniki) povsem zadostni in dovolj sposobni za opravljanje inteligentne akcije. Tega ni mogoče dokazati, trdi Feigenbaum, ker je to delovna hipoteza. Če v to verjamete, ste delavec na tem področju. Če v to niste prepričani, pač niste umetni inteligenčnik. Feigenbauma tudi ne zanimajo sterilne razprave o tem, ali je strojna inteligenca teoretično mogoča in ali se koncepti umetne inteligence dotikajo etičnega ozadja. Eden takih kritikov je npr. Joseph Weizenbaum z MIT, ki govori o nevarnosti umetne inteligence zaradi nedopustnega poenostavljanja pri razpoznavanju človekovega uma (v knjigi Computer Power and Human Reason, 1976).

Feigenbaum pravi, da je Weizenbaumova kritičnost neodgovorna in napačna. Weizenbaum se vsiljuje kot svetnik in virtukrat (uradnik kreposti). Raziskovalci na področju kognitivnih znanosti, ki jih kritizira, spadajo med najpomembnejše znanstvenike dvajsetega stoletja: da so to velikani, je danes očitno. Tu dosežejo Feigenbaumovi izbruhi svoje vrelišče. Predpostavlja, da je Weizenbaum s svojo paničnostjo zgrešil bistvo. Mentalnost je opredeljena z več dimenzijami, vendar za večino teh dimenzij ni interesa. Weizenbaum se vprašuje, ali stroji lahko čustvujejo, toda zakaj bi investirali denar v takšen odgovor. Na skrajnem koncu spektra umetne inteligence se zanimanje za človekov um pojavlja enostavno zaradi umske lokacije inteligence. Ljudje umetne inteligence verjamejo, da obstaja taka množica inteligenčnih principov, ki velja neglede na to, ali se ta inteligenca pojavlja v ogljikovih ali silicijskih sistemih. Ti ljudje iščejo principe in ne človeške kvalitete, vendar je edini ustrezno dostopni inteligentni sistem prav humanoid. Pri tem velja upoštevati še to, da je odkrivanje neprimerno lažje od invencije in da so današnji sistemi umetne inteligence zaradi tega tudi usmerjeni v posnemanje človekovih sposobnosti. Tesna povezava umetne inteligence in kognitivne psihologije pa je izredna.

Feigenbaum tako nepopusttljivo zavrača Weizenbaumovo obtožbo, da umetna inteligenca omalovažuje status človekovega uma. Model človeka kot informacijskega procesorja ne pomeni razvrednotenja človeka. Ta model povečuje človekovo veličino, njegovo razvojno stopnjo, ko dopušča uporabo lastnega umskega mehanizma za razlago, kako sam človek deluje. To je prav gotovo velik izziv, s katerim se soočamo. In kaj odseva ta refleksija, kaj bi um lahko bil? Feigenbaum odgovarja s strojem, brez dvoma in brez omahovanja. Strojna inteligenca je že tu.

Post scriptum. Zanimivo bi bilo vedeti, kaj bi lahko postoril Feigenbaum, če bi mu kako "dobro stoječe" slovensko podjetje dalo proste roke pri oblikovanju vodstvenega (predvsem finančnega) ekspertnega sistema. Tudi če bi npr. odpadle vse težave s srednjim vodstvenim slojem in s šefi informacijskih centrov, bi moral Feigenbaum s svojo ekipo šele odkrivati zakonitosti anarhičnega in nepredvidljivega finančnega kompleksa slovenskega podjetja. In verjetno bi s to svojo izkušnjo lahko bistveno obagatil in posplošil svoje sedanje znanje, ki je usmerjeno predvsem na dokaj regularne in dobro identificirane pojavnosti ameriškega poslovnega prostora. Mogoče pa bi mu naši umetni inteligenčniki lahko nekaj takega predlagali. In lahko bi se zgodilo, da bi Feigenbaum tako ponudbo sprejel in napisal novo knjigo v sodelovanju z našimi strokovnjaki: The Rise of the Expert Company under the Economically Unforeseeable Circumstances. To pa bi lahko postal pravi priročnik in uspešnica ekonomistov in podjetnikov nerazvitega (zlasti neuvrščenega) sveta in Feigenbaumov osebni doprinos k reševanju svetovne krize planetarnega juga.

Anton P. Železnikar

Intel's ISSCC Bombshell: A Supercomputer on a Chip

At this year's International Solid State Circuits Conference in New York Intel has introduced 80860, a 40-Mhz chip that packs more processing power per square micron than anything in its class this year. It is a reduced-instruction-set-computer chip that is no less than the silicon equivalent of a Cray 1 supercomputer.

The 80860 delivers 33 million VAX instructions/s running the Stanford test suites, 10 million floating-point operations/s on the Linpack benchmark, 90,000 Dhrystones, and 500,000 4-by-4 vector transforms/s performing a graphics operation. In addition, it has been designed for multiprocessor operation, so the company has multiprocessor Unix development under way to support it. Intel will start selling the 80860 before the year is up for $750 in 1,000-unit lots.

The chip comes with a RISC integer CPU, single- and double-precision pipelined vector floating-point processors (a separate adder and multiplier), 3-d graphics, 4-Kbyte instruction and 8-Kbyte data caches, and paged memory-management unit. It uses a 64-bit data and instruction bus to execute more operations per second. With such a wide data path, the computer can load a 32-bit integer and 32-bit floating-point instruction, which execute concurrently in a single clock cycle.

Moreover, because the unit has separate multiplier and adder, a multiply-and-accumulate operation can occur in a single clock cycle as well.

The graphics processor can operate on bytes of data, a powerful feature for handling color. In one instruction cycle, the unit can individually process the values to drive the red, green, and blue guns of a color monitor and still have a byte operation to spare. But raw processing power is useless unless processors can rapidly acquire data to feed these units.

The 80860 has a system of buses that could not be implemented easily on a printed-circuit board with discrete processor chips. A 64-bit bus feeds instructions to both integer and floating-point processors. It transfers 320 Mbytes/s of data. Another bus runing between the data cache and the floating-point unit is 128 bits wide and can move 640 Mbytes/s. The external bus to the DRAMs is 64 bits wide and can transfer data at 160 Mbytes/s.

There are special instructions for performing pipeline accesses of large data structures, that don't completely fit inside the cache, called pipelined floating-point load, in which the data is brought directly into the floating-point hardware without going through the on-chip cache. The pipeline floating-point load instruction of a large data structure implies a one-time use of the data coming into the floating-point unit; so it is not desirable to flush the cache and load it with information that will only have to be flushed afterward anyway. This feature - preventing a cache flush and loading data directly From RAM at full speed by overlapping memory accesses from the three banks of memory - provides a factor-of-three speed increase.

Three-dimensional graphics capability was added after the initial chip-architecture definition. It occupies only 3% more silicon area, to give about 10-times performance improvement in 3-d graphics.

Intel has recognized the importance of providing software for the 80860. In introducing the chip in late February, the company announced a complete set of development tools including C and Fortran compilers and Fortran vectorizers. Intel also has an early prototype of a multiprocessing version of Unix on the 80860, now being upgraded to system V, Release 4.0.

(Electronics, March 1989) Hitij

Deset principov strategije industrijskega računalniškega podjetja

Uvod

Kako začeti strateško načrtovati razvoj industrijskega računalniškega podjetja? Ali je mogoče postaviti deset osnovnih razvojno-smiselnih in izkustveniuh principov?

Induktivno mišlenje o strategiji je že nekaj časa vodilni princip upravljavske inteligence uspešnih podjetij. Tržna usmerjenost in izhodišča za uresničevanje visoke tehnologije so strateški elementi na področjih elektronske industrije, v katero sodita tudi komunikacijska in računalniška industrija. Tudi pri nas se pojavlja industrija, ki postaja čedalje pomembnejša proizvajalka in prodajalka na profesionalnem in konzumnem računalniškem tržišču. V zadnjih desetih letih se je podoba domače računalniške industrije radikalno spreminjala in se v kriznih razmerah spreminja povsem nepredvidljivo. Zaradi tega ne bi kazalo napovedovati njene prihodnje strategije in taktike, ki bi premočno temeljili na njeni preteklosti.

Doslej se je naša računalniška industrija razvijala pretežno na domačem trgu, dokler ni dosegla kritičnega stanja v svoji organiziranosti in poslovni podjetnosti. V tem trenutku kritična masa domače računalniške industrije ni razvojno, kadrovsko in poslovno konsolidirana. Npr. profitnost poslovne usmerjenosti naj bi se opredeljevala nanovo, trženje bi se moralo razširjati na več tujih trgov, raziskave in razvoj naj bi povečale industrijsko moč v smeri večje kakovosti in količine produktov, novo strateško mišlenje in "naravni" upravljavski koncepti so potrebni pri ugotavljanju nujnosti republiških, federalnih in še posebej tujih vlaganj in industrijske kooperacije itd.

Osnovni principi strateškega mišljenja domače računalniške industrije

(1) Domača računalniška industrija (DRI) bo povečevala obseg svoje prodaje do 1,2 milijarde ameriških dolarjev v desetih letih in bo letno dosegala 15 odstotni ali večji dobiček z nadaljno rastjo produktivnosti.

(2) Profitnost DRI bo temeljila v izbiri njenih produktov, v kateri bodo supermini in mini računalniki in PC/profesionalni mikroračunalniki.

(3) Velika priložnost DRI so lahko bistvene spremembe v republiški in federalni zakonodaji in možnosti vključevanja v Evropsko skupnost. V tem kontekstu bo DRI povečevala svoje napore pri vlaganjih, kooperaciji in podjetniškem združevanju, z nadaljevanjem vrste fundamentalnih sprememb, ki so se že začele.

(4) DRI bo opravila dva pomembna tehnološka preboja, ki bosta vsak po svoje bistveno spremenila prihodnost njene lastne in spremljajoče industrije. Ta preboja sta Računalniško integrirana komunikacija (angl. CIC) in Računalniško integrirana proizvodnja (angl. CIM).

(5) DRI lahko napreduje na več frontah. Močan pritisk se bo pojavil v mednarodnih vlaganjih in v okviru evropske industrijske kooperacije. Oboje lahko razširi trg DRI ne samo v industrijski avtomatizaciji, temveč tudi na področjih CAD/CAM, visoko zmoglivih in znanstvenih naprav.

(6) DRI bo izboljševala svoje razumevanje in sprejemanje mednarodnih (zlasti evropskih) industrijskih standardov in metodologij in ne bo podpirala lokalnih in subkulturnih standardov in metodologij.

(7) DRI bo podpirala strategijo Odprtega trga s ciljem utrjevanja prodaje lastnih in kooperacijskih produktov. Dobršen del prodaje DRI bo izvajan tudi s prodajalci, ki niso neposredni uslužbenci DRI.

(8) DRI bo vlagala v majhna, inovativna podjetja, ki bodo obvladovala tako aparaturno kot programsko opremo in tista, ki ji bodo lahko hitro pomagala pri dopolnjevanju njenih zahtev po racunalniško integrirani komunikaciji (CIC).

(9) Tako imenovano pisarniško in tovarniško tržišče bosta ostali glavni usmeritvi marketinške strategije DRI. To pa ne pomeni, da DRI ne bo razvijala nekaterih posebnih tržnih in produkcijskih aktivnosti.

(10) Skladno s pomembnostjo lastnega razvoja bo DRI skrbno spremljala ravnovesje med njenim trženjem in tehnološkimi dejavnostmi na zapadnih in vzhodnih trgih in pri industrijski kooperaciji.

Povzetek

Ta zapis predpostavlja, da bo trenutna strategija DRI omogočala doseganje ciljev, ki so bili zapisani v odstavkih prejšnjega poglavja. Glavni cilji DRI so rast v okviru industrije, mednarodni industrijski standardi in metodologije, konkurenčni produkti in usluge in trajna profitnost. Ti cilji bodo dosegani z mednarodnim ravnovesjem pri vlaganjih in industrijski kooperaciji, z ustrezno industrijsko integracijo, alternativnim marketingom, odprtimi sistemi, razvojnimi programi CIC in CIM.

Nemudoma je nujno zgraditi takšno marketinško organizacijo, ki bo sposobna generirati zadosten obseg naročil, potrebnih, da se lahko produkcijske zmogljivosti DRI vzdržujejo na profitni stopnji in da bo omogočeno razširiti prodajne dejavnosti zlasti na zapadnih trgih.

Pričujoči zapis je predvsem začetek razvoja bojnega načrta DRI, in to korak za korakom. Odtod je mogoče izpeljevati podrobnejše strateške (planske) in taktične (operativne) zamisli.

A. P. Železnikar

Transputer – mikroprocesor, ki prihaja

Tržišče 32-bitnih mikroprocesorjev se je v zadnjem letu povečalo za 230%. Intel in Motorola sta še vedno najmočnejša mikroprocesorska prozvajalca. Nov, prodorni mikroprocesorski proizvod pa je transputer, čeprav je njegova prodaja v Evropi leta 1988 dosegla šele $18m, naj bi to tržišče v letu 1992 znašalo že $400m. Tudi največji računalniški prizvajalci in zlasti proizvajalci delovnih postaj ponujajo svoje riscovske računalnike. Trendi v smeri RISC tehnologije naraščajo zaradi možnosti proizvodnje superhitrih delovnih postaj, miniračunalnikov in t.i. vmeščenih krmilnikov. Pojav transputerja je prekinil dominacijo Intela in Motorole. V letu 1992 naj bi tržišče RISC mikroprocesorjev doseglo že tretjino tržišča visokozmogljivih mikroprocesorjev. Uporaba RISC procesorjev naj bi naraščala stitikrat hitreje kot uporaba navadnih mikroprocesorjev. Trenutno je registriranih vsaj 12 različnih tipov RISC procesorjev.

V čem je prednost RISC (Reduced Instruction Set Computing) v primerjavi z CISC (Complex Instruction Set Computing)? RISC ima precej hitrejši izvajalni mehanizem kot CISC zaradi manjše ukazne zaloge. En ukaz lahko obdela v enem samem ciklu stroja. Tudi veliko število teh enostavnih ukazov se lahko procesira hitreje kot se procesirajo zaporedja ekvivalentnih kompleksnih ukazov v CISC procesorju. Celo IBM je začel z intenzivnim razvojem RISC conceptov in prijavil že več patentov za posamezne transputerske dele in tehnologijo. IBM je uzakonil RISC s svojo delovno postajo že pred tremi leti. Hewlett-Packard se je spustil v tvegano igro tako, da je svojo linijo Spectrum v celoti opremil z RISC procesorji. RISC je tako postal najresnejši izziv proti dominantni mikroprocesorski arhitekturi, ki sta jo razvila Intel in Motorola. Trg pa čedalje bolj potrjuje sprejemljivost RISC tehnologije. Tako se tudi Intel pojavlja s svojim RISC mikroprocesorjem 80960, ki naj bi bil predviden za uporabo vmeščenega krmilnika. Večina RISC procesorjev je optimizirana na uporabo operacijskega sistema Unix, ki je na področju uporabe delovnih postaj zelo popularen. Ob tem pa se RISC tehnologija pomika čedalje očitneje v poslovne aplikacije in se ne omejuje več le na specialno inženirsko uporabo.

Vendar sam RISC procesor ne bo dovolj. Aplikacije RISC procesorja zahtevajo posebne periferne krmilnike. Motorola že ima enoto s plavajočo vejico v svojem procesorju 88000. Ker

je RISC arhitekturno enostavnejši od CISC, je tudi cena na enoto lahko znatno nižja. Industrijski analitiki računajo, da bo ta cena padla na $1000 za 1MIPS v letu 1992. V RISC je pravzaprav vključenih tudi vrsta lastnosti CISC. Za Motorolin 88000 se poudarja zmožnost njegove uporabe pri razširjenem paralelizmu.

Masovno vgrajevanje RISC mikroprocesorjev v procesne enote računalnikov bo znižalo ceno na enoto zmogljivosti. Ta masovnost pomeni paralelno arhitekturo računalnikov. Tu velja poudariti, da je hitrost integriranih procesorjev povezana tudi z njihovo toplotno disipacijo, tj. s problemi odvajanja toplote oziroma s povečano ceno hlajenja. S paralelizmom pa se bo enormno povečala zmogljivost računalnikov, tako da bo najbrž odklenkalo današnjemu konceptu dragih glavnih računalnikov in superračunalnikov. Ta proces paralelizacije bo pospešen zlasti zaradi padanja cen mikroprocesorjev.

Tržišče paralelnega procesiranja se šele oblikuje in vanj že vstopajo tudi največji računalniški proizvajalci. Samo malo, neznano podjetje Alliant Computer je v lanskem letu prodalo za $75m svojih supermini sistemov. Ti računalniki imajo zmogljivosti supersistemov ob cenah minisistemov, ko uporabljajo paralelno arhitekturo s svojimi integriranimi procesorji. Za paralelne arhitekture pa je seveda potrebna tudi paralelna programska oprema, npr. inteligentni kompilatorji. Privlačna so tudi paralelizirana jedra operacijskega sistema Unix. S takšnim pristopom je pri povezavi 30 procesorjev mogoče doseči zmogljivost sistema 120 MIPS (npr. računalnik podjetja Sequent). Transputer je prav za prav osnova za masivne paralelne procesorje. S transputersko arhitekturo je mogoče zmogljivosti sistema enostavno prilagajati potrebam, hkrati pa je ta arhitektura skrajno modularna. Tako je mogoče s transputerskim konceptom gospodarno pokriti kar najširši spekter zmogljivosti, npr. od osebnega računalnika s 5 MFLOPS do zahtevnega sistema z 200 MFLOPS. Pri velikem številu procesorjev mora biti tudi pomnilnik porazdeljen. Seveda pa mora ostati zunanji vtis, da je pomnilnik globalen, ker je splošno računalniško tržišče še slabo seznanjeno s stroji, ki uporabljajo porazdeljeni pomnilnik. Pri sistemih z globalnim pomnilnikom se pojavljajo resne težave, če je število procesorjev večje od 30. Transputer kot procesor podpira klasično, pomnilno in komunikacijsko procesiranje in je zato pripraven kot gradnik paralelne procesorske mreže.

Osnovni problem vseh paralelnih sistemov pa ostaja prej ko slej programska oprema.

A. P. Železnikar

Od ptičjega petja do nevrogeneze

Novica, ki jo objavlja časopis Scientific American (februar 1989), je lahko zelo zanimiva tudi za konstruktorje nevralnih računalnikov. Potrjuje namreč, da se v naravi že dogaja to, kar so si upali najdržnejši konstruktorji le domnevati. Zakaj pravzaprav gre?

Pri študiju središč za krmiljenje petja v možganih kanarčka so odkrili, da se rojevajo nove živčne celice v zreli dobi življenja in da lahko nadomestijo stare celice. Takšna nevrogeneza kaže na možnosti, kako bi bilo mogoče dosegati samopopravljanje (avtoreparaturo) tudi v človekovih možganih. Seveda pa bi želeli imeti to lepo lastnost smaovzdrževanja tudi v nevralnih računalnikih.

Eno izmed najtrdnejših prepričanj nevroznanosti je bilo doslej, da se vsi nevroni ali živčne celice oblikujejo v dobi dozorevanja, ko tudi možgani rastejo. Verjelo se je, da ima odrasli vretenčar fiksno število nevronov. Tako naj bi se nevroni, ki so bili uničeni zaradi bolezni ali poškodb ne nadomeščali in učenje se naj ne bi razvijalo z nastajanjem novih celic v nevralnih mrežah, ki krmilijo vedenje, temveč le z vzpostavljanjem povezav med omejenim številom nevronov.

Nova evidentnost nevrogeneze se je pojavila na povsem nepričakovanem področju: pri raziskavah učenja petja pri ptičih. Pri tem so opazili, da se ne rojevajo stalno le novi nevroni v ptičjih možganih v dobi ptičje zrelosti, temveč novo rojeni nevroni zamenjujejo tudi postarane nevrone. Ti novi nevroni naj bi predvsem prevzeli novo informacijo, ki nastaja z učenjem. Sposobnost učenja petja pri mladih in odraslih ptičih naj bi bila tako odvisna predvsem od pojavljivosti novih nevronov, ki lahko oblikujejo nove mreže. Pri tem pa bi morda bilo mogoče odkriti, kaj bi lahko pripeljalo do razkritja faktorjev, ki bi stimulirali človekove možgane, da bi popravljali same sebe z nadomeščanjem poškodovanih nevronov z novonastalimi.

A. P. Železnikar

Ekološke raziskave poklicnega programiranja

Že več kot dvajset let raziskujejo psihologi vidike programske produktivnosti in kakovosti z opazovanjem konstruktorjev in uporabnikov programske opreme. Pri tem uvajajo sistematične mere vedenja v procese razvoja programov in v raziskave novih metod programiranja in tehnologij z odpiranjem novih socialnih in kognitivnih interpretacij programirnih procesov. Psihologi zato tudi sodelujejo pri oblikovanju in definiciji novih programskih artefaktov. Ekološko oblikovanje programske opreme si tako postavlja nove oziroma svojske norme in cilje.

Programiranje je velikokrat skupinsko delo. Tradicionalni model vodnega slapa pri razvoju programov z natančnimi specifikacijami navzdolnjega razvoja dobiva še nove prijeme, ki poudarjajo hitro snovanje in iterativno izboljševanje programa. Pri tem so pomembni zlasti skupinska organizacija, skupinski procesi, politika vodenja, možnosti ponovne metodološke uporabnosti, razvojna orodja, oblikovalne metode, strategije odkrivanja napak in vzdrževanje programov. Ti faktorji pomenijo t.i. novo programirno paradigmo, ki pa je še vedno v svoji otroški fazi. Postavljanje oblikovalnih zahtev in razvojnih prototipov namreč ne spada med standardne dejavnosti psihologov. Ta vloga psihologa pri razvoju programov šele nastaja. Nova paradigma zahteva tudi razvoj idej, ki neposredno vplivajo na programirno produktivnost in kvaliteto.

Oblikovanje programov je dejavnost, ki se dogaja na delovnih mestih izven psiholoških laboratorijev in je predvsem tehnološko terminirano. Potrebna je izčrpnejša analiza

poklicnih programrjev v okolju realističnih nalog. V študiji posameznega primera se analizirajo podrobnosti, kvalitativna informacija, intervjuji. Tako zbrani popdatki so lahko zelo bogati in odpirajo različne perspektive.

Kot že povedano, spadata gradnja in invencija artefaktov v področje, ki v raziskovalni psihologiji ni standardno. Sodobna programska oprema je prepletena s psihološko vsebino, npr. v razpravi o racionalnosti strukturiranega programiranja ali o objektno naravnanem programiranju. Tu naj bi šlo za sistematizacijo psihološke vsebine pri tovrstnem razvoju programov. Pojavlajajo se t.i. projekti ekološkega oblikovanja programov. V njih se uvaja pojem nalogovno-atrefaktnega cikla: psihološke analize nalog, ki jih programerji radi opravljajo, povezanih s problematiko, videnjem in zadovoljstvom njihovega izkustva; temu sledi postavljanje ciljev za nova programirna orodja, ki pa so omejena s tehnološkimi zmogljivostmi. Nova orodja dejansko spreminjajo naloge, za katere so bila oblikovana, spreminjajo situacijo, v kateri so naloge nastopale in spreminjajo celo pogoje, zaradi katerih so jih programerji želeli uporabljati. Vse to pa zahteva nadaljno analizo nalog in čas za izdelavo novih artefaktov.

Ekološke oblikovanje je še vedno nastajajoči pristop, v katerem naj bi psihološki razmisleki vodili oblikovanje novih programskih artefaktov v realnih situacijah. Ključni problem teh projektov je namera, da se integrirata psihološka analiza naloge v realni delovni situaciji in razvoj novih programskih artefaktov, s katerimi se izboljšujejo uporabnost, produktivnost in zadovoljnost. Pri tem je pomembna tudi psihološka analiza vloge in uporabe dokumentacije in njene oblike pri programskem vzdrževanju.

Pomembne so npr. tudi izkušnje, kako se študenti učijo programiranja v lispu in kako delujejo sistemi za poučevanje tega jezika. Tak sistem-učitelj je lahko v regularni uporabi v kurzih o jeziku lisp in je običajno uspešnejši od standardnega poučevanja. Vloga kognitivnih in sociooloških znanosti pri invenciji programske tehnologije je lahko očitno izredno pomembna. Morda bi bil čas, da bi tudi v našem delovnem okolju sprožili podobno raziskavo. Časopis Communications of the ACM (november 1988, letnik 31, številka 11) je objavil več člankov o ekologiji programiranja.

A. P. Železnikar

Finančne in razvojne novice

Nekatere finančne novice so lahko zanimive tudi za domačo računaniško industrijo, če začenjamo v okviru nje razmišljati tudi s pogledom na mogoč prihranek in dobiček. Prav zaradi tega smo v tem pregledu izbrali finančne podatke manjših podjetij, ki dosegajo velikost domačih razmerij.

Podjetje Emulex, ki proizvaja vtične plošče za DECove računalnike, poroča o polletnem dobičku $7,3m, povečanem za 84%, pri prometu $73,8m. Evropsko tržišče je zadovoljivo, izjavlja predsednik podjetja Fred Cox.

Podjetje Mentor Graphics, producent delovnih postaj, poroča o dobičku v letu 1988, ki znaša $33,5m, povečanem za 65%, pri prometu $221,8m, povečanem za 36%.

Podjetje za bančno programsko opremo Hogan Systems je povečalo devetmesečni dobiček za 48,9% na $1,5m kljub dodatnim stroškom $4,6m zaradi suspenzije nekega razvojnega projekta in redudančnih plačil. Dohodek je narastel za 4,1% na $35,2m.

Zanimiva je dohodkovna rast podjetja Intel od njegove ustanovitve v l. 1968 do leta 1988: 1968=$2,6k; 1969=$565,6k; 1970=$4,2m; 1971=$9,4m; 1972=$23,4m; 1973=$65,5m; 1974=$134,4m; 1975=$136,7m; 1976=$225,9m; 1977=$282,5m; 1978=$399,3m; 1979=$660,9m; 1980=$854,5m; 1981=$788,6m; 1982=$899,8m; 1983=$1,1bn; 1984=$1,6bn; 1985=$1,3bn; 1986=$1,2bn; in 1987=$1,9bn.

Glavne tehnološke inovacije (zanimive za vse tiste, ki so ta razvoj doživljali tudi na svoji koži pri nas) pa so bile tele:

1969 64-bitni bipolarni RAM; 256-bitni MOS SRAM;
1970 1k MOS DRAM;
1971 2k EPROM; 4-bitni mikroprocesor;
1972 1k MOS RAM; 8-bitni mikroprocesor;
1974 mikroprocesor 8080;
1975 8k EPROM;
1976 4k SRAM; mikroprocesor 8048;
1977 16k EPROM;
1978 32k EPROM; mikroprocesor 8086;
1979 16k DRAM; mikroprocesor 8088;
1980 16k E-EPROM; mikrokontroler 8051; koprocesor 8087; 16-bitni mikroprocesor;
1981 32-bitni mikroprocesor; 64k EPROM;
1982 128k EPROM; 64k DRAM; mikroprocesor 80186; mikroprocesor 80286;
1983 256k EPROM;
1984 512k EPROM; 64k CMOS EPROM;
1985 256k CMOS EPROM; mikroprocesor 80386;
1986 megabitni EPROM;
1988 vmeščeni krmilnik 80960.

Podjetje Compaq obrača že dve milijardi dolarjev (konec leta 1988 že $2,066bn). S tem je podjetje verjetno doseglo stanje, ko se začne razvojna krivulja umirjati. Samo še IBM in Apple prodata več PCjev kot houstonsko podjetje, ki je prodalo samo v ZDA v lanskem letu pol milijona naprav. Čisti dobiček je znašal $255m, kar je 87% več kot leta 1987. Največji uspeh je bil dosežen na trgu naprav s procesorjem 80386.

Intel nima avtorske pravice za procesorja 8088 in 8086. Pravda Intela z japonskim podjetjem NEC, ki prodaja Intelova plagiata z oznako V.20 in V.30, je bila na ameriškem sodišču zgubljena. Vzrok za tako odločitev sodnika je napaka Intela, ki ni pravilno zaščitil svojih mikroprocesorjev.

A. P. Železnikar

Analogni nevralni procesor podjetja Fujitsu

Japonsko podjetje Fujitsu je razvilo nevralni procesor v BiCMOS tehnologiji za splošno uporabo. Posamezno integrirano vezje deluje kot živčna celica (nevron). Nevralni procesor (nevroprocesor) je predviden za obdelavo analognih signalov. S povezavo posameznih vezij (nevronov) ga je mogoče uporabiti tudi za paralelno obdelavo signalov. Nevralni podsistem procesorja zmore 80000 operacij multipliciranja in seštevanja v sekundi. Uporaba nevroprocesorja je mogoča v sistemih umetne inteligence, senzorjih za analogno obdelavo podatkov in v robotskih krmilnih sistemih v pogojih realnega časa.

A. P. Železnikar

Nevralna vezja tudi v tovarni

V okviru evropskega raziskovalnega združenja Esprit so odprli projekt za industrijsko uporabo nevralnih vezij v evropskih industrijskih podjetjih. Sodeluje 10 podjetij in raziskovalnih organizacij iz štirih držav. Projekt z vrednostjo $6m naj bi še utrdil položaj raziskav nevralnih vezij nasproti ZDA, saj je zaostanek minimalen. Namen projekta je razvoj programirnih orodij in demonstacija zmogljivosti nevralnih vezij. Projektu načeljuje Harwellski laboratorij britanske atomske komisije. Med udeleženci projekta pa so nemški Siemens in Tehniška univerza Darmstadt, britanski British Aerospace in Artificial Intelligence, francoski raziskovalni center za mehaniko Cetim in dve grški podjetji za raziskave nevralnih vezij, in sicer Alpha in NTU. V SFRJ se ukvarjajo s takšnimi raziskavami tudi na strojniški fakulteti v Beogradu. Le kdaj se bodo podobnim projektom lahko pridružili naša federalna vlada in kakšna naša institucija?

A. P. Železnikar

BrainMaker: simulator nevralnih vezij

BrainMaker je program za simulacijo nevralnih vezij, ki omogoča kreiranje, gradnjo, vadbo, preizkušanje in izvajanje nevralnih vezij. S programom je mogoče odločati o sestavi in specializaciji vezja. Prizvajalec trdi, da lahko BrainMaker izvaja do 500000 nevralnih povezav v sekundi in da podpira štiri tipe linearnih in nelinearnih nevronov. Program nudi tudi vhodno/izhodno manipulacijo visualnih ali simbolnih podatkov. Priloženi so vzorčni programi za optično razpoznavanje karakterjev govorno sintezo, razpoznavanje slik in slikovno povečevanje.

Program zahteva IBM PC, XT, AT, PS/2 ali kompatibilca, 256k zlogov RAMa, DOS 3.0 ali višji DOS in monokromatski ali barvni prikaz.

Cena: $99,95.

Proizvajalec: California Scientific Software, 160 East Montecito, Suite E, Sierra Madre, CA 90124.

EUREKA sproža skupni programirni projekt

Trinajst evropskih podjetij iz petih držav - Francije, Norveške, Švedske, Združenega kraljestva in Zapadne Nemčije - se povezuje v projektu Eureke z namenom, da se aktualizira evropsko znanje na področju programske opreme in okrepi kompetitivnost na svetovnih tržiščih. Udeleženci nove Eurekine programske tovarne (Eureka Software Factory - ESF) nameravajo izboljšati produktivnost programskega razvoja, izdelati evropske standarde in razviti programsko knjižnico. Ta desetletni projekt bodo podpirale posamezni vlade in udeležena podjetja z zneskom $420m, sedež projekta pa bo v Zapadnem Berlinu. Dvig programirne produktivnosti postaja nujen problem, saj so že sedaj zahteve večje od razvojnih zmogljivosti. V Zapadni Evropi raste trg programske opreme 16% letno in naj bi leta 1992 dosegel produkt $45b. Eureka (the European Stratetig Program for Research and Development in Information Technology) je panevropska pobuda. Med udeleženci ESF so francoska Matra, norveški Norsk Data, švedski Telelogic, britanski ICL in zapadnonemška podjetja AEG, Softlab in Nixdorf Computer.

Morda bi se temu projektu lahko priključila tudi SFRJ s podjetjem Iskra Delta. Splačalo bi se vsekakor poskusiti.

Anton P. Železnikar

Nastajanje mlade informatike

Tudi v Nemčiji je informatika na univerzah zašla v napetostno polje javnosti in gospodarstva. V marcu 1989 se je vrenje zaradi nezadovoljstva pojavilo na nemških univerzah v obliki štrajkov in protestnih shodov na področju celotne zvezne republike. Kakšni so vzroki tega nezadovoljstva informatikov? Štrajki so se pojavljali že konec leta 1988. Novo študentsko gibanje je ideološko neobremenjeno zahtevalo spremembe zaradi katastrofalnega stanja izobraževanja. Profesorji v prenapolnjenih predavalnicah čedalje bolj vztrajajo na zastareli učni snovi in se posebej ne ozirajo na vmesna vprašanja, pripombe in sporne strokovne podrobnosti. V seminarjih in praktikumih delajo študentje brez motivacije s pomočjo asistentov, ki morajo zaradi teh obveznih vaj zanemarjati svoje raziskovalno delo. Računalniška opremljenost je količinsko in kakovostno nezadostna. Pojmi delovne postaje in komunikacijske mreže niso uresničeni. Programska oprema je antikvarna in se dopolnjuje le z lastnim razvojem. Poklicna orodja ali sistemi, ki si za prakso pomembni, na univerzah sploh ne obstajajo.

Na te pomanjkljivosti se navezuje tudi vsebina študija, ki je teoretsko omejena in se ne

ozira na prakso. Ne obstajajo tudi interdisciplinarne institucije, ki bi lahko gledale preko roba svoje stroke. Prav tako ni institutov za prakso, ki bi študenta pripravili na njegovo poklicno delo. Zgolj z denarjem in osebjem obstoječega stanja ne bo moč izboljšati. Vzroki mizerije nemške informatike (tako se pri njih uradno imenuje študij računalništva) so globoki. Univerze se nahajajo v napetostnem polju med gospodarstvom, javnostjo in družbo. Prav v informatiki pa se to stanje jasno kaže z vso surovostjo. Pričakovanja se morajo spremeniti.

Post scriptum. Po vsem tem lahko ugotovimo, da so razmere na nemških univerzah podobne razmeram na obeh slovenskih univerzah. To je lahko cinični argument politike, gospodarstva pa tudi samih fakultetnih delavcev, da je neperspektivno stanje pri nas mogoče še poslabšati.

A. P. Železnikar

IBM sprejema mrežni standard OSI

IBM je končno sprejel računalniški mrežni standard OSI (Open System Interconnection) in uvedel dva nova programska produkta, ki olajšujeta mrežam in računalnikom komunikacijo s produkti drugih proizvajalcev. Za standardom OSI stoji npr. tudi britanska vlada. Medtem se je pojavilo na trgu še 50 drugih programskih produktov za uporabnike računalniških mrež na IBMovih velikih računalnikih. To omogoča izmenjavo podatkov med akademskimi in vladnimi organizacijami. Takšen produkt je npr. razširjena verzija IBMovega NetView.

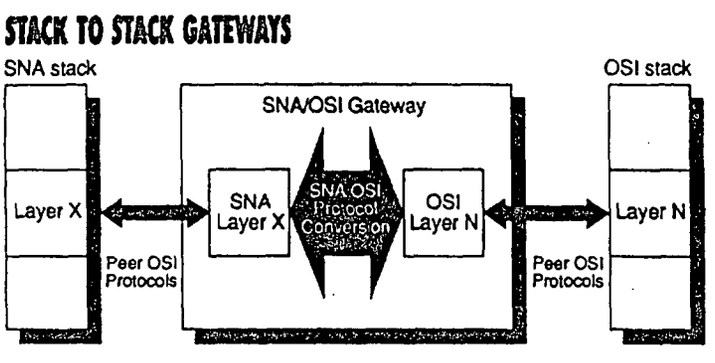
IBM je že doslej uporabljal prožno strategijo prodaje mrežne opreme, in sicer v ZDA svoj SNA (System Network Architecture) v Evropi pa je podpiral OSI standarde, npr. od leta 1985 ravnini 4 in 5). Seveda bo IBM prodajal svoje mrežne produkte, dokler bo to mogoče. Vendar je najbrž na mestu trditev, da bosta morala SNA in OSI doseči status koeksistence. Na problemu povezave teh dveh sistemov delajo tudi druga podjetja. IBMovi produkti, ki podpirajo OSI produkte so npr. še OTSS (Open System Transport and Session Support), OSNS (Open System Network Support), PROFS (povezava tipa X.400 za Pofessional Office System), DISOSS (Distributed Office Support System) in MTA (Message Transfer Agent tipa X.400).

Razlika med OSI in SNA uslugami pa je tale:

Protokolske usluge	OSI	SNA
emulacija terminala	protokol virtualnega terminala (VTP)	3270 LU 2.0
upravljanje mreže	CMIS/CMIP	NetView
zbirčni prenos in dostop	FTAM	ECF (Enhanced Connectivity Facility
imenski servis	X.500 Directory Service	4/5/2.1 SNADS
elektronska pošta	X.400 Message Handling System	SNADS
NetBIOS	TOP Standard on Class 4 Transport	IBM on 802.2 Data Link Control APPC Alternative
medmrežni protokoli	CLNS Network ES-IS Route Exchange	Source Routing

Pomen kratic v tabeli je tale: APPC = Advanced Program to Program Communication; CLNS = Connectionless Network Services; CMIS/CMIP = Common Management Information System and Protocol; ES-IS = End System-Intermediate System; LU = Logical Unit; PU = Physical Unit; SNADS = Systems Network Architecture Distribution Services. Naslednja slika kaže portal med skladoma sistema SNA in OSI.

A. P. Železnikar

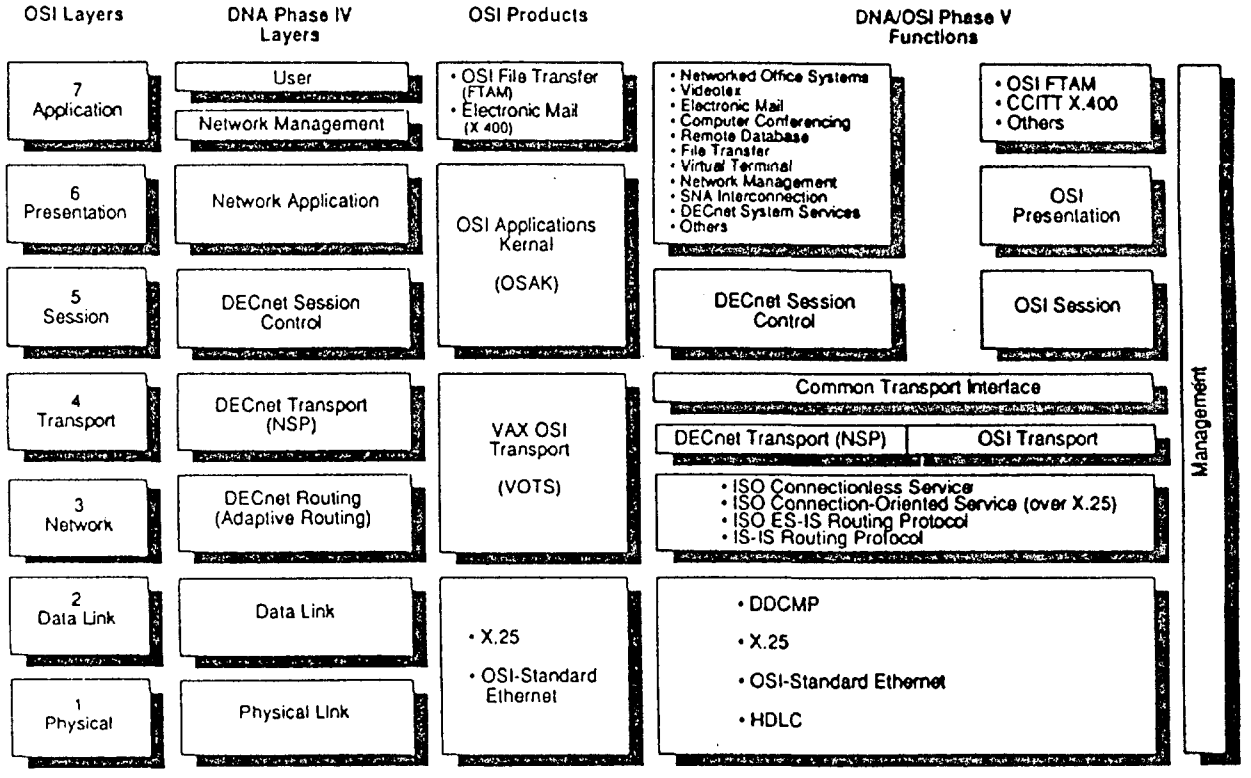


DEC se pomika v komunikacijsko integracijo

Ameriški proizvajalec miniračunalnikov Digital Equipment Corporation (DEC) napoveduje svoj mrežni (komunikacijski) program za podjetniško upravljanje, skupaj z nemškim Siemensom, ki naj bi proizvajal t.i. module dostopa. Ti moduli naj bi omogočali upravljanje v večuporabniški mreži. To seveda ni ničesar drugega kot premik v smeri OSI. Vzporedno s tem naj bi se uporabljal tudi produkt imenovan Podjetniške usluge, s funkcijami konzultacij iz planiranja, načrtovanja, implementacije in upravljanja. Celoten posel naj bi prinesel DECu dohodek do $11bn v treh letih. Tudi Evropa je močno DECovo tržišče. DECovo migracijo iz njegove DNA (Digital Network Architecture) v smeri DNA/OSI že v letu 1990 kaže shema na sliki.

Fazni pomik DECa v področje OSI ima tako nekaj specifičnih produktov:

VAX FTAM zadeva prenos zbirk (File Transfer), dostop (Access) in upravljanje (Management) protokolov in omogoča zbirčno upravljanje in prenos k OSI sistemom. VAX FTAM omogoča uporabnikom kopiranje, brisanje, prenos in prikaz zbirčne informacije med odprtimi siste-



mi. Pri tem ohranja semantiko in zbirčne podatke, ki so bili izmenjani.

Portal poročilnega slednika X.400 je usluga za elektronsko pošto, ki zadeva protokolni standard X.400 in omogoča komunikacijo med obstoječim DECovim pošt nim sistemom in drugimi sistemi tipa X.400. Poštni slednik seže v DECov MAILbus, ki povezuje uporabnike ALL-IN-1 s poštnim sistemom, kot je IBMov PROFS (Professional Office System) in SNA Distribution Services (SNADS).

OSAK (OSI Application Kernel) je polna implementacija OSI ravnine zasedanja, ki omogoča uporabo programirne povezave (API) ali knjižničnih rutin, ki urejajo dialog med sistemi OSI mreže.

VOTS (VAX OSI Transport Service) omogoča uporabo funkcij OSI ravnine prenosa. VOTS podpira sodelovanje med različnimi podmrežami z uporabo Internet protokola, ki nudi usluge nepovezane mreže prek lokalne mreže in daljinskih mrež tipa X.25.

A. P. Železnikar

Philips vstopa v proizvodnjo dinamičnih ramov

Največji evropski proizvajalec polprevodnikov Philips vstopa v posel proizvodnje dinamičnih RAMov zaradi nevarnosti, ki jo predstavlja japonski delež, ki znaša že 50%. Ta posel naj bi bil realiziran v naslednjih dveh letih. Siemens samplira trenutno 4Mbit DRAMe in ta tehnologija naj bi bila prenešena tudi v Philips pod okriljem skupnega projekta Megaproject. Philips ima dobre poslovne zveze tudi s podjetjem Matsushita EC, v katerem ima 35% delež za 35 let in pogodbo o tehnološki izmenjavi. Matsushita pa je vodilno podjetje na trgu 4Mbitne tehnologije.

Philips je sedmo največje polprevodniško podjetej na svetu, ki pa doslej ni izdelovalo pomnilniških integriranih vezij. Šele v lanskem letu je stekla proizvodnja EPROMov v obsegu $50m, letos SRAMov in v letu 1990 še DRAMov. Tako naj bi bil sčasoma kompenziran manjko na trgu, ki omejuje zlasti proizvodnjo PCjev in znižanje njihovih cen.

A. P. Železnikar

Optična rešitev

Raziskovalci Londonske univerze in podjetja Philips so konstruirali polprevodniške sendviče za optične računalnike. Našli so učinkovit način za pretvorbo elektronskih v optične signale, kar pa je izredno pomembno za razvoj optičnih računalnikov. Najprej so izdelali silicijske optične detektorje, ki pretvorijo optične signale v elektronske, kar je bil lažji del naloge. Obratna pretvorba pa je težji problem. Sedaj se raziskujejo silicijske naprave s t.i. polprevodniško plastjo III-V, ki raste pri vrhu. Silicij je nosilec logike, snov III-V pa pretvarja elektronske signale v svetlobo. Doslej so pokazali več naprav, ki so rastle v galijevem arzenidu na silicejevi podlagi.

A. P. Železnikar

Vojaške raziskave postajajo komercialne

Britansko obrambno ministrstvo je odločilo, da bodo vojaški raziskovalni centri odslej morali poslovati komercialno. Ti centri zaposlujejo 12500 uslužbencev v skoraj 100 ustanovah. Smatra se, da je teh uslužbencev preveč in da bodo odslej ti centri sami morali nositi poslovno tveganje. V enotno agencijo bodo združeni signalne in radarske naprave, letalstvo ter raziskave admiralitete in oborožitve. Agencija bo sprejemala raziskave iz civilnega sektorja in iz prekomorskih dežel. Ta reorganizacija je del splošne politike vlade, ki želi povečati operacije vrednost-za-denar za civilne usluge.

A. P. Železnikar

Računalništvo kot disciplina

Poseben odbor ACM je oblikoval izhodišča za učni načrt računalništve kot discipline. To aktivnost predlaganja učnih programov vzdržuje ACM že 42 let. T.i. računalniške znanosti niso le programiranje, temveč precej več, npr. tudi načrtovanje aparaturne opreme, sistemska arhitektura, oblikovanje plasti operacijskega sistema, strukturiranje podatkovnih baz za specifično uporabo in modeli vrednotenja. Poudarjanje programiranja nastaja iz našega dolgoročnega prepričanja, da so programirni jeziki sijajni pripomočki za osvajanje preostanka discipline. To prepričanje pa omejuje našo sposobnost, da bi govorili o disciplini v vsej njeni širini in bogastvu.

Priporočilo, ki je nastalo v odboru pod vodstvom Petra J. Denninga, daje t.i. jedro možnega programa. Opis računalništva kot discipline sestavljajo štirje deli: zahteve, kratka definicija, razdelitev na podpodročja in elaboracija podpodročij. Kratka definicija je tale: Računalništvo obsega kot disciplina sistematičen študij algoritmičnih procesov za opisovanje in transformacijo informacije, in sicer: teorijo, analizo, oblikovanje, učinkovitost, implementacijo in uporabo. Fundamentalno vprašanje celotnega računalništva je, "Kaj bi bilo mogoče (učinkovito) avtomatizirati?"

Polje računalništva naj bi se pokrivalo z devetimi podpodročji, in sicer:

1. algoritmi in podatkovne strukture
2. programirni jeziki
3. arhitektura
4. numerični in simbolni izračuni
5. operacijski sistemi
6. programirna metodologija in tehnika
7. podatkovni in informacijski sporočevalni sistemi
8. umetna inteligenca in robotika
9. komunikacija človek-računalnik

Te discipline naj bi obsegale teorijo, abstrakcijo in oblikovanje (snovanje, načrtovanje) glede na razpoložljivi učni čas. Pri tem naj bi se poudarjalo za posamezno disciplino značilno mišljenje, ki daje sposobnost za invencijo novih distinkcij, ki vodijo do novih načinov akcije in novih orodij, ki omogočijo, da te distinkcije postanejo dostopne tudi drugim. Pomembna je tudi uporaba orodij: gre za sposobnost uporabe orodij področja pri učinkovitih akcijah v drugih domenah. Seveda mora biti zagotovljeno tudi organizirano laboratorijsko delo za študente.

A. P. Železnikar

O globalni logiki strateških koalicij

Podjetja se učijo tega, kar je nacijam znano že od pamtiveka: v zapletenem in negotovem svetu, v katerem se pojavljajo nevarni nasprotniki, ni dobro živeti sam. Zveza (antanta) je tedaj vselej odgovorno delo dobrega strateškega repertoarja. Kakšna naj bi bila normalna strateška koalicija? Seveda so mogoči priložnostni skupni posli (joint venture) pa tudi dolgoročna pogodbena razmerja. Toda napredujoča zveza s temi načini se le redkodaj lahko vzdržuje. Prava zveza namreč kompromitira osnovno neodvisnost ekonomskih akterjev in direktorji je ne marajo. Za direktorje pomeni upravljanje (poslovanje) totalno kontrolo, zveze pa to kontrolo delijo.

V stabilnih kompetitivnih okoljih je alergija zaradi izgube vpliva dokaj nepomembna. Na hitro se spreminjajočih globalizirajočih trgih in v industriji, v svetu konvergirajočega ukusa potrošnikov, hitro napredujoče tehnologije, naraščajočih fiksnih stroškov in protekcionizma pa temu ni več tako. Globalizacija opravičuje zveze, ki postajajo bistveni elementi strategije. Razvoj, ki se uvršča pod naslov globalizacije, zahteva zveze (antante) kot nujnost. Za tekmovanje v globalni areni se je potrebno izpostavljati in pokrivati ogromne fiksne stroške: torej potrebujete partnerje.

Marsikdo bi si danes želel živeti in nakupovati v Kaliforniji. Tam si želijo najboljše produkte po najnižjih mogočih cenah. Svetovna informacija omogoča neglede na nacionalnost, da si potrošniki želijo podobne vrste produktov ali celo podoben način življenja. Ekonomski nacionalizem se razkazuje v volilnih kampanjah, okužuje volilno telo in oblikuje skupinske interese. Neglede kje živite - v Evropi, Japonski ali ZDA - ukus potrošnikov se lahko izenačuje ne le pri določenih ekonomskih vrednostih, temveč tudi v vrednotah. Npr. slovenski Elan osvaja ZDA in Japonsko, nemški Adidas Formozo itd. Tržišče podjetja IBM ni več odvisno od državnih ali zemljepisnih meja. Fujitsu ima izključno prvenstvo v svojih strojih za numerično obdelavo, in sicer s 70% na Japonskem in s 50% v svetovnem merilu.

Današnji produkti temeljijo na vrsti kritičnih tehnologij, ki jih večina podjetij ne more samostojno obvladovati. Poslovna programska oprema, ki je pripomogla ibmovskemu PCju do neslutenega uspeha, ni bila IBMova. Npr. program 1-2-3 je proizvedlo podjetje Lotus Development Corporation. Operacijski sistem za PC je izdelal Microsoft, integrirana vezja pa Intel. V globalni strategiji je čas proizvoda lahko celo pomembnejši od njegove tehnologije. Tehnologija je tudi vselej začasna. Le dober igralec lahko zmojstri vse elemente uspešnosti. To pa pomeni, da sodeluje s partnerji in s tem profitno plasira tudi svoje produkte.

Imeti nadzor nad vsem, ne pomeni nujno boljše poslovanje. Z dovolj časa, denarja in sreče je mogoče vse postoriti samostojno. Toda kdo ima danes vsega tega dovolj? Konvergenca tržnih potreb je skupaj z neizprosno razpršenostjo tehnologije spremenila poslovno logiko, s katero morajo direktorji krmariti svoja podjetja. V preteklosti je bilo mogoče marsikaj dosegati z monopolom. Globalizacija je klasični monopolizem izničila, ker je postal enostavno preveč tvegan zaradi možne konkurenčnosti podjetij z globalno logiko. V današnjih razmerah poslovanja je potrebno vselej iskati pot zmanjševanja ogromnih fiksnih stroškov. Igra spremenljivih stroškov praktično ni več mogoča. Partnerji so potrebni za ublažitev fiksnih stroškov in skupaj z njimi je potrebno opredeljevati strategije, ki maksimalno prispevajo k vašim fiksnim stroškom.

Morala tega nauka je očitna. Ko je avtomatizacija izrinila delovno vsebino iz produkcije, je proizvodnja kmalu postala dejavnost s fiksnimi stroški. In ker so se stroški razvoja prelomnih idej in njihove uresničitve v tržne produkte hitro povečevali, so tudi raziskovalno-razvojni oddelki postali fiksen strošek. Če je npr. v farmacevtski industriji potrebnih $50m ali več za razvoj novega zdravila, RR niso več igra spremenljivih stroškov. Z globalizacijo so lahko ali lahko postanejo vsi glavni igralci v vaši industrijski branži vaši direktni konkurenti. Pri tem je več ali manj neznano, ali bodo konkurenti želeli z vami deliti neko specifično tehnologijo. Zaradi vsega tega so partnerji potrebni, toda tudi sami potrebujete lastne ljudi in lastne laboratorije. To pa so fiksni stroški.

Tudi izdelava in vzdrževanje novega izdelka je fiksen strošek, še zlasti njegova zaščita na trgu. Zaščita je brez pomena, če izdelek na trgu ni prodrl. Tudi polovično izvedena promocija novega izdelka je slabša kot nobena. Včasih je nove izdelke bolje kar prodajati prek prodajne mreže, saj se s tem neposredno ne povečujejo fiksni stroški. Pomik proti fiksnim stroškom v prodaji in v distribucijskih mrežah je v zadnjem razdobju vse bolj viden. Igra spremenljivih stroškov je možna z uporabo drugih prodajalcev, toda le do določene stopnje. Toda vaša prodajna delovna sila še vedno mora opravljati vzdrževanje, podporo produktov, šolanje in pisanje priročnikov. Vse to pa so fiksni stroški. Nekaj teh stroškov je lahko še vedno spremenljivih. Najamete lahko ceneno delovno silo, pomaknete proizvodnjo v dežele v razvoju, toda vse to danes ne bo več zadostovalo. V preteklosti so lahko bili stroški spremenljivi z uporabo deljenega časa računalnikov in informacijskih poslovnih sistemov. Toda izkušnja je pokazala, da principa deljenega časa ni mogoče uporabljati, če želite imeti sistem za vaše lastne potrebe, torej sistem, ki vam lahko nudi konkurenčne prednosti. Tako je danes tudi informacijska tehnologija postala fiksen strošek. V daljšem razdobju pa seveda vsi ti fiksni stroški lahko postanejo spremenljivi z uglaševanjem investicijskih ravnin (kapitalnih izdatkov). Toda kratkoročno so to fiksni stroški.

V povedanem je tedaj fundamentalna sprememba, ki je nastala v zadnjih desetih do petnajstih letih. Fiksni stroški zahtevajo novo logiko v aktivnosti direktorjev. V spremenljivih stroških se skriva povečevanje profita, in sicer z zmanjševanjem stroškov materiala, plač, delovnih ur itd. Nova logika sili direktorje v amortizacijo fiksnih stroškov preko celotne tržne baze, vse to pa kaže v smer globalizacije.

Direktorji morajo opustiti lažno prepričanje, da totalna kontrola povečuje možnosti uspeha. Nevarna je aritmetika, ki pravi, da je 51% deleža treba povečati na 100% in 49% znižati na 0%. Že 51% zagotavlja vašo polno legalno kontrolo. Toda to je tudi kontrola aktivnosti na tujih trgih, o katerih veste le malo, če sedite daleč od njih. Američani in Evropejci, ki pridejo na Japonsko, želijo vselej imeti 51% deleže. To je magično število, ker zagotavlja prednostno pozicijo pri kontroli osebja, tržnih odločitvah in pri izbiri investicij. Toda dobro partnerstvo - podobno kot zakonska zveza - ne deluje na temeljih lastnine in nadzora, zahteva pa napor, zaupanje in entuziazem na obeh straneh, če si želi uspeh. Ko se sčasoma partnerstvo poglobi, lahko postanejo ločene entitete nesmiselne. Strategija, vrednote in kultura se lahko tako ujamejo, da je možna nova kombinacija. Hewlett-Packard je leta 1963 začel na Japonskem skupen posel s podjetjem Yokogawa Electric v razmerju 51-49%. Po dveh desetletjih zaupanja je Yokogawa Electric odstopil podjetju Hewlett-Packard nadaljnih 24%. Trajalo je dve desetletji, da je HP dosegel znatno lastninsko pozicijo. Tako ima HP danes 75% $750m podjetja.

Zveze so tedaj kot zakoni - delujejo le, če delujeta oba partnerja. Zveze niso formalne pogodbe. Pri njih ne gre za nakup in prodajo pravičnosti. Oba partnerja verjameta da imata edinstvene spretnosti in funkcionalne zmogljivosti glede na drugega. In oba morata biti ves čas marljiva, da bi bila zveza uspešna. Če je eden od partnerjev slab in len in nepripravljen za napor, da bi izrabil skupne prednosti, se lahko zveza razdre. Enostranskost in asimetrija naporov in pozornosti uničujejo medsebojno razmerje.

Anton P. Železnikar

talking on tuesdays

torkovi pogovori

Razvojni credo Slovenije in Iskre Delte

Anton P. Železnikar (4. aprila 1989)

Uvod

Z besedo credo ne bi želel manipulirati. Izbral sem jo zavestno in preudarno. Ta beseda, ki pomeni latinsko molitev pri maši z začetkom, me spominja na izpoved ob nekem začetku, ki naj bi bil začetek drugačnega, seveda uspešnega razvoja. Torej ne credo quia absurdum, ki je vera v nesmiselno, temveč credo ut intelligam, ki je vera, da bi vsi skupaj spoznali kam in kako.

Še vedno sem na samem začetku, kjer nizam argumente. Slovenija je pripravila svojo izpoved za področje računalništva in informatike in v tem republiškem nareku sem našel po svojem prepričanju tudi kaj uporabnega, čeprav sem se jezil nad sociologično diagnostiko preteklosti, ki mi je vzela predvsem dragoceni čas. V tem nareku je vsaj za moj zahtevnejši ukus zmanjkalo tega, kar bi glede na svojo življensko izkustvo lahko poimenoval s politično kirurgijo. Japonci podobne državne listine o svojem razvoju novih računalniških generacij ne bi napisali v slogu, ki se že na samem začetku zgubi v analitiki in naštevanju, ki za novo pot v prihodnost domala nista prav nič pomembna. To svoje izhodišče bom poskušal diskurzivno utemeljiti še kasneje.

To, kar je na republiški ravni spodbudno, je ustanovitev Komisije Evropa 92 na predlog Republiškega komiteja za mednarodno sodelovanje. Cilj te komisije je vključevanje slovenskih političnih in gospodarskih subjektov v EGS. Svoja predstavnika v tej komisiji imata tudi Iskra in Gorenje. Treba bi bilo posebej pogledati, kako se področje računalništva in informatike lahko uveljavi v delu te komisije.

Ob ponovni izvolitvi za direktorja podjetja Iskre Delte je Janez Škrubej zapisal svoj razvojni credo podjetja za obdobje naslednjih štirih let (1989 - 1993). V tem zapisu je gotovo marsikaj razvojno uporabnega, nekatere bistvene razvojne možnosti pa so zamolčane: tiste, ki npr. zadevajo povečevanje kapitalnih prihrankov in profitnosti, avtomatizacijo proizvodnje s tržno utemeljitvijo in seveda tudi to, kar bo morda v naslednjem štiriletju najpomembnejše, kapitalno, investicijsko, tehnološko in kadrovsko integracijo v Evropo.

S projekcijo neke strategije podjetja sem se tudi sam ukvarjal in konec lanskega leta sem jo tudi izpisal kot dokument za lastno rabo, predvsem pa kot začetek na poti pisanja in učenja neke resnejše, šele nastajajoče ali možne strategije. Oceni o profitnosti se nisem mogel izogniti, razvoj pa sem postavil tudi v desetletno obdobje. Informatizacijo in robotizacijo podjetja sem skopo zajel v okviru pojmov CIC in CIM, ki sta za ta tip podjetja v svetovnem prostoru neizbežni. Poudaril sem usodnost zakonodaje in povezave v Evropo, morda dovolj resno le v svoji glavi in premalo izrazito v samem dokumentu.

Še vedno sem uvodno razpoložen. Vendar bom sedaj začel soočati tri poglede na razvoj računalništva in informatike: republiškega oziroma državnega, podjetniškega, kot izhaja iz zapisa Janeza Škrubeja in svojega. S tem se ne izmikam nikakršnji odgovornosti in dopuščam vsakršen ugovor, ki je dovolj strokoven, vendar tudi pluralen in demokratičen. Prav takšen način soočanja se mi dozdeva danes v industriji potreben, da se z njim lahko aktivirajo intelektualni potenciali podjetja, da se prežene strah pred argumetirano in javno besedo. S tem predgovorom to soočanje tudi začenjam.

Republiška analitika in začetki neke strategije

Dokument, ki npr. očrtuje prihodnji razvoj informatike v Sloveniji ima naslov "Informacijska tehnologija v strategiji in politiki razvoja SR Slovenije". O tej strategiji je razpravljal samo vrh republiške SZDL. Nikomur ni prišlo na misel, da bi ta dokument ponudil v kritično razpravo npr. Slovenskemu društvu Informatika, ki je član SZDL ali pa mogoče celo sindikalni organizaciji, ki pokriva podjetja z dejavnostjo računalništva in informatike. Tako daleč uradovalna pamet zaenkrat resda še ne seže.

Ta republiški dokument ugotavlja, da je informacijska tehnologija pomembna zadeva, tako pomembna za družbeno infrastrukturo, da mora postati skrb za njo tudi državna. V zvezi s tem ugotavlja, da je sam dokument tudi že strategija, čeprav je v svoji zasnovi bolj analitičen in nekako prepričevalski kot pa usmerjen v nujno radikalnejšo aktivnost. Dokument postavlja v neposredno razmerje t.i. napredek reprodukcije (s tem misli tudi na industrijski, razvojni, družbeno infrastrukturni napredek) in informacijsko tehnologijo. Podzavestno postavlja tedaj splošno enačbo

napredek = informacijska tehnologija

Ta enačba se mi zdi otroško naivna zaradi vrste razlogov, ki ne spadajo neposredno in ne le v ta kontekst. Pojmovanje informacijske tehnologije razširja, ne samo na računalniško materialno in programsko opremo, temveč tudi na človeka, na njegove cilje in vrednote. S tem propagira seveda informacijski humanizem oziroma humanistični informacionizem, ki ga poskuša partikularizirati s pojmom informacijske pismenosti. Kot da človek na današnji stopnji svoje evolucije ni predvsem informacijsko bitje v samem sebi in v socialni integraciji, oziroma se tega ne zaveda, vsaj dovolj prepričljivo ne.

To kar republiški dokument omenja kot razvoj strategije do leta 1984, se mi zdi bolj posle-

dica spontanega vedenja različnih akterjev - družbenih in posameznih. Dosežki v razvoju industrije, šolstva, komunikacij, morda tudi raziskovanja itd. so bili predvsem spontani. Politični posegi so bili uradniški in niso bistveno premikali stanja na boljše, npr. Zakon o zadevah posebnega družbenega pomena na področju računalništva. Pri vsem tem je šlo bolj za dovoljevanje določenega obnašanja kot pa za usodne posege politike v informatiko ljudstva in države. Danes je mogoče ugotoviti, da politični vpliv, ki bi lahko bil tudi strateški, ni pustil opaznih ali spodbudnih posledic niti v informacijski industriji, računalniških znanostih in informacijski uporabi. Od zgodnjih osemdesetih let pa do danes smo priča le navideznim vzponom (npr. Iskre Delte), praviloma pa razkroju in zmedi, ki se še danes stopnjujeta.

Npr. število osebnih računalnikov bi v Sloveniji še bistveno bolj naraslo, če ne bi bilo carinskih, davčnih in uvoznih limitov. Če se danes sklicujemo na te številke (npr. 10000 PCjev v Sloveniji), potem moramo povedati, da večjega števila državljanom nismo omogočili. Računalniško pismenost smo tako predvsem bistveno omejevali, ne pa podpirali. To omejevanje so občutili osebno tudi učitelji, ki se spontano niso uspeli izobraziti za poučevanje računalniških predmetov v šolah. Podobno ugotovitev bi lahko postavili tudi za uslužbence, ki se sami niso mogli izobraževati zaradi pomanjkanja osebnih računalniških učil.

V Raziskovalno skupnost Slovenije je informatika kot interdisciplinarno in posebno znanstveno področje vstopala kot novinec in velikokrat kot pastorek. Prav zaradi tega so jo lahko posvajali tudi nestrokovnjaki (diletanti, prenapeteži) v svojih matičnih strokah. Nekateri obrobni problemi pa so se lahko s silo avtoritet tudi napihovali. Republiški dokument je marsikje to napihovanje povsem nekritično prenesel v svojo terminologijo. S tem je sporne zadeve aktualiziral, namesto da bi jih postavil v objektivni kontekst. Vendar je dokument prinesel tudi zanimiv podatek o finančnem deležu raziskav, ki pokrivajo informatiko, tj. 11%, s predlogom, da se ta delež poveča na 20%. Tako bi relativno nominalno povečanje glede na druge programe RSS zneslo celih 73%. Mogoče si je seveda zamisliti, kaj vse bo naredila kraljica znanosti fizika in kako se bo prelevila v informatiko, da bo svoja visoka sredstva lahko obdržala neokrnjena. Le upamo lahko, da se iz tega ne bo izcimila nova politična igra.

Republiški dokument ugotavlja, da se informacijska administracija (tj. tista, ki je podprta z IT) uveljavlja v turističnem, transportnem in bančnem gospodarstvu in v sistemu republiške uprave. Zlasti navaja telekomunikacije oziroma javno omrežje, kjer omenja JUPAK, dotika pa se informativno tudi integriranega sistema podatkovnih mrež prihodnosti (ISDN). Republiški dokument prav gotovo napihuje pomembnost projekta COSINE, kjer gre za standardno (domala že rutinsko) povezavo univerzitetnih mrež, ki nima posebne industrijsko standarizacijske oziroma splošno družbene infrastrukturne relevance.

Odstavki, ki so namenjeni informacijski industriji v Sloveniji, so v dokumentu zgolj analitični in ugotavljajo znana dejstva o industrijski revoluciji, strukturi Iskre Delte (strojna oprema tipa DEC in IBM, princip OEM), mikroelektroniki, elektrooptiki in o odpiranju vzhodnoevropskih tržišč. Bistvena je pripomba o pomanjkanju stategije vključevanja v svetovna tržišča in zlasti v Evropo, ko industrija ni realizirala pridobivanja tujega kapitala, razvojnega sodelovanja in proizvodnje zahtevnejših informacijskih sistemov. Pri tem dokument le sramežljivo govori o potrebnosti dodatnih razvojnih sredstev za pospeševanje razvoja informacijske industrije.

Dokument upravičeno in poudarjeno artikulira problematiko informacijske standarizacije, ko se sklicuje na ustanove ISO (mednarodni standardi), ISO TC 97 (računalništvo in informatika), CCIIT (telekomunikacije in še posebej dokument X.200, ki zadeva ISDN) in IEC (elektrotehnika). V okviru vsega tega izpostavlja OSI (integracijo odprtih sistemov). Ob tem dokument verjetno le premalo upošteva pojavnost t.i. de facto standardov, ki postanejo dostikrat zmagovalci v uporabniški, tehnološki in stroškovni tekmi z institucionaliziranimi vladnimi standardi.

Slovenija naj bi končno dozorela za vstop v t.i. informacijsko družbo. To je seveda zaenkrat lahko le deklaracija po želji možnega razvoja. To svojo deklarativnost razbija tudi sam dokument, ko govori o organizacijskem infantilizmu in tehnološki paranoji - tudi v območju računalništva in informatike. Strinjam se, da je socialna paradigma razvoja še posebej za kompleksno področje IT in njene uporabe potrebna, kjer je del te paradigme tudi spodbujajoča politika republike. Pomisleke pa imam s t.i. razvojnimi scenariji, ki niso kaj več kot izolirani diskurzi in prikrojevanje k partikularističnim interesom neformalnih preostankov dejanske moči vplivanja v zaprtih administrativnih okrožjih. Industriji je predvsem potrebna akcija države, ki omogoča uspešnost (profitnost, rast) informacijske industrije, ne pa njeno prekomerno obremenjevanje, npr. z vzdrževanjem nepotrebne družbene infrastrukture (npr. neprimernega odvzemanja in razporejanja akumulacije, vzdrževanja fundamentalne psevdoznanosti itd.).

Vstop Slovenije v informacijsko družbo naj bi bil omogočen z naslednjo aktivnostjo: izdelavo in uzakonjenjem državne strategije razvoja; usposabljanjem in aparaturnim opremljanjem mednarodno konkurenčnih raziskovalno razvojnih skupin v industriji, malih podjetjih in selektivno tudi na univerzi; mednarodno konkurenčnostjo industrijskih produktov; informacijskim izobraževanjem; prestrukturiranjem t.i. čiste raziskovalne dejavnosti (RRS) v korist informatike (s povečevanjem števila raziskovalcev, s finančno prerazdelitvijo od 11% na 20% ali če več); industrijskimi konzorciji za razvoj IT; konzorciji za razvoj gospodarske infrastrukture (PTT, železnica, energetika, republiška uprava itn.); javnimi naročili pri domači računalniški industriji; izvozom domače računalniške industrije na tržišča razvitih dežel; in utrjevanjem položaja domače informacijske industrije s t.i. razvojnim dinarjem in vključevanjem v svetovno produkcijsko mrežo.

Ta zadnji del republiškega dokumenta je brškone nabit z možno aktivnostjo, ki bi se je morali tudi z industrijske plati lotiti z vso zagrizenostjo in brez popuščanja. To pa zahteva najbrž tudi stalno dopolnjevanje in posodabljanje lastne strategije Iskre Delte, predvsem pa tudi višjo stopnjo njene domiselnosti - torej polnejšo izrabo njenega intelektualnega (poslovnega, organicacijskega, finančnega, marketinškega, proizvodnega, servisnega in razvojno

tehnološkega) potenciala.

Post scriptum. Republiški dokument o strategiji razvoja IT je preobsežen in nesistematičen, težko berljiv in razumljiv le za nekoga, ki zna sam povezati bistvene razvojne elemente preteklosti in sedanjosti. Dokument se preveč in nepotrebno ukvarja z analitiko preteklosti ob bistveno spremenjeni sedanjosti. Strategija naj bi bila načrt za prihodnost. V dokumentu je tudi opazna sentimentalna navezanost na znanost kot na tisto realsocialistično doktrino, ki že sama po sebi pogojuje in zagotavlja uspešno razvojnost neglede na to, v kakšni tehnološki in poslovni nesposobnosti ta znanost domuje in ko zaradi svoje socialne gloriole ni povezana s preživetjem populacije, ki to znanost vzdržuje. Od republiškega razvojnega dokumenta bi pričakoval predvsem sistematično razdelavo tega, kar je zapisano v njegovem zadnjem poglavju z naslovom Razčiritev strategije družbenega razvoja. Dokument pa naj bi pregledala tudi strokovna društva informatikov, ekonomistov, organizatorjev, upravnih delavcev, itd. in podjetij, institutov in fakultet, ki se jih ta dejavnost dotika. Ob vsem tem pa bilo modro podpirati tudi javno in strokovno polemiko.

Strategija računalniške industrije

Že v uvodu svoje programske usmeritve Iskre Delte (v novembru 1988) se Janez Škrubej sklicuje na Belo knjigo EGS, ki je temelj tega, kar naj bi z letom 1992 nastalo v Evropi kot enotno tržišče. Omenja tudi Kompleksni program znanstveno tehničnega sodelovanja SEV kot odgovor na Eureko. Temu dodaja podatke o predvideni rasti zapadne in vzhodne ekonomije.

Iskra Delta je v razdobju od leta 1984 do 1988 znatno investirala v šolski center v Novi Gorici, v svoje zapadno operativno zastopstvo v Avstriji, v izgradnjo produkcijskega centra v Stegnah pri Ljubljani, ki ima industrijska obrata še v Titovem Velenju in na Ptuju ter v konsolidacijo svojih delovnih enot na ozemlju SFRJ. Avtor programske usmeritve trdi, da bo s to lastninsko, kadrovsko in organizacijsko bazo mogoče dosegati cilje, ki jih našteva v desetih strateških izhodiščih.

Programske usmeritve J. Škrubeja so delno strateške in delno že operativne. So primer značilne mešanice vizije, trenutnih pogojev gospodarjenja in preostankov preteklosti. V začetku leta 1993 naj bi Iskra Delta pri 2000 zaposlenih dosegla bruto produkt $450m, od tega 50% za izvoz. To bi zneslo $225k per capita. Ta podatek je seveda spodbuden, zahteval pa bo bržkone kaj več, kot je zapisano v programskih usmeritvah, saj bi povečanje per capita znašalo kar 4,5-krat tega, kar naj bi bilo doseženo v letu 1989. Žal v tej bistveni finančni točki strategije manjka podatek o predvidenem doseganju in rasti profitnosti, ki je temelj povečevanja operativnosti in razvojnosti podjetja.

Druga točka programske usmeritve je namenjena proizvodnemu programu, tj. proizvodnji lastnih in kompatibilnih produktov (DEC, IBM, Honeywell), komunikacijam (CIC) in industrijski kooperaciji. Produkti Iskre Delte pokrivajo tako industrijo, turizem, trgovino, energetiko in bančništvo (točka 3). Naslednji točki poudarjata konstantno raven zaposlenosti (2000 uslužbencev), ustanavljanje kooperativ, štipendiranje najboljših in kariero uslužbencev. Šesta točka poudarja relikt preteklosti, tj. nagrajevanje delavcev po rezultatih dela in administrativno spremljanje stroškov. Informacijski sistem Iskre Delte (npr. CIC, CIM in ekspertni sistem za tako vodenje podjetja, ki povečuje kapitalne prihranke in profitnost) je predmet sedme točke, v naslednji točki pa se poudarja uravnovešenost med zahodnimni in vzhodnimi tržišči. Zadnji točki programske usmeritve pokrivata šolanje uporabnikov doma in v tujini in mednarodno stadarizacijo.

Sklep Škrubejeve strategije je optimističen in spodbuden, saj govori, da bo Iskra Delta že leta 1992 sodobno organizirana, mednarodno standarizirana in povezana, primerno profitna in utemeljena na znanju njenih uslužbencev. Skozi to svojo usmerjenost naj bi se vključila v krog vodilnih računalniških podjetij Evrope.

Post scriptum. Republiški dokument razvoja informatike bi lahko na določnejši način podpiral Škrubejevo strategijo. Značilno je tudi, da se Škrubej pravzaprav ne zanaša ali ne verjame v neko blagodejno osveščenost in pomoč političnega sistema Slovenije pri razvoju informacijske industrije v ožjem, tj. računalniškem pomenu. S strategijo, ki jo je Škrubej predložil in jo uslužbenci Iskre Delte de facto podpirajo (Delavski svet, vodilni in strokovno kvalificirani uslužbenci in bržkone tudi večina uslužbencev) nastaja tudi obveza, da se glavni finančni cilj v štiriletnem razvoju res doseže. Hkrati pa se morajo upoštevati še nekateri strateški faktorji, ki so bili zaradi posebnih razvojnih težav v Škrubejevem dokumentu zamolčani (npr. profitnost, struktura podjetniškega informacijskega sistema, konkretnejše vključevanje v Evropo, nova motiviranost, itd.).

Poudarki iz zasebne strategije

V začetku oktobra 1988 sem napisal dokument z angleškim naslovom "Principles of Strategic Thinking of Iskra Delta Computers" z namenom, da njegov način izražave postane paragon ekspozitorne jasnosti (kondenzirane relevantne vsebine) in posebne jezikovne konciznosti. (V angleščini sem tedaj ta posel lažje opravil kot bi ga v slovenščini.) Bil sem prepričan, da mora dokument za strateško rabo podjetja vsebovati določeno simboliko, privlačnost in nedorečenost, ki prizadetemu še vedno omogočajo individualizacijo oziroma operacionalizacijo naporov v različnih položajih.

Dokument, ki je izhajal iz dolgoročne strategije podjetja, si je za desetletni cilj postavil brutoprodukt $1,2bn. Pri tem sem uporabil primerjavo z rastjo najvidnejših evropskih podjetij računalniških produktov in cilj, da postane Iskra Delta z njimi konkurenčno primerljiva. Zanimivo je, da si je podoben cilj že leta 1980 postavil Ivan Atelšek, tedanji generalni direktor Gorenja, vendar v tistih časih gospodarska in politična elita tega še nista bila sposobni doumeti. V svojem dokumentu sem izpostavil še tele elemente: povprečno dobičkonosnost 15%, produkcijo superminijev, minijev in PC profesionalnih mikroračunalnikov, ustavne spremembe in vstop v EGS, uvedbo CIC in CIM, mednarodne investicije in evropsko industrijsko kooperacijo, privzem mednarodnih standardov in dušitev subkulturnih, strategijo odprtega trga, investiranje v mala podjetja, razvoj tako pisarniškega kot tovarniškega tržišča in ravnovesje med vzhodnimi in zapadnimi tržišči. Ti principi naj bi bili uresničljivi s predpostavko, da je Iskra Delta že v sedanjem položaju sposobna generirati zadosten obseg naročil, ki

bi omogočal profitno vzdrževanje lastnih produkcijskih kapacitet, seveda ob razširitvi poslovanja na zapadna razvita tržišča.

Iz te vsebine bi danes še enkrat izpostavil elementa CIC in CIM, ki sta oba po svoji naravi računalniška in sta operativna osnova vsakršnje računalniške industrije, tj. njene profitnosti in možnosti razvoja.

Sklep

Prispevek obravnava različne vidike razvoja računalništva in informatike: državnega, industrijskega in infrastrukturnega. Državni vidik naj bi upošteval strategijske standarde razvitih držav, vključno z industrijo in uporabniško (v najširšem pomenu) infrastrukturo. To naj bi bil hkrati standard državniške kvalitete. Industrijski vidik naj bi upošteval kvalitativne strateške poudarke (tudi njihovo razkrivanje) uspešnih srednjevelikih (izjemoma tudi velikih) industrijskih računalniških podjetij (Apple, Nixdorf, Olivetti, Siemens, DEC, IBM). Osebna strategija uslužbenca v tej igri pa naj bi bila tista strateška sinteza, ki omogoča nastajanje novih strategij - državnih in industrijskih - skozi zasebno in javno pobudo.

Grafični standardi v luči uporabniške povezave

Andrej Terčelj

Grafika pomeni ključ k diferencijaciji proizvodov.

Mladost računalniške industrije je klasificirala računalniško grafiko v dve veji. Prvo predstavljajo računalniški standardi, ki jih potrjujejo organizacije kot sta ISO in ANSI in so dolgotrajni in težko dosegljivi procesi. To skupino predstavljata GKS in PHIGS kot primera, kjer je prikazana razvojna kronologija od zametka do končne potrditve. Ob tem so bile navedene in razložene tudi ovire pri sprejemanju standardov in glavne kratice, ki se pojavljajo v poplavi grafike od PC do CAD/CAM okolja. Drugo stran pa predstavljajo "de facto" gstandardi, ki so po definiciji učinkoviti, saj se potrjujejo na tržišču. Tu prevladuje IBM kot postavljalec in uveljavitelj standardov. CGA, EGA in VGA grafične kartice predstavljajo IBM standard medtem ko Hercules Technologies proizvajalca NE IBM standarda.

Končni cilj grafičnih standardov je nadomestitev krmilnih programov tako, da ne bodo vezani več direktno na specifično aparaturno opremo. Določene rešitve ponujajo proizvajaci v PostScriptu in DPS-Display PostScriptu in različnih uporabniških vmesnikih, ki pa z grafičnimi standardi še nimajo iste vloge in definicije. Na koncu sem primerjal še različne grafične postaje, ki že ali še bodo prevladovale v devetdesetih letih. To so SUN v NeWs ,NeXT z DPS in NeXT Windowing, UNIX grafične postaje z X Windows in Triglav z GKS jedrom.

Kako so nastali torkovi pogovori

Torkovi pogovori so nastali na pobudo Vanje Bufona kot strokovni informativni forum Poslovne enote za razvoj, raziskave in innovacije Iskre Delte, na katerem teče pogovor o najnovejših dosežkih računalništva in informatike po svetu in v podjetju, in sicer z vidika tehnologije, uporabe in znanosti. Opravljena sta bila le dva torkova pogovora, saj so bili kasneje ti pogovori preneseni na petek.

Prvi pogovor (14. marca 1989) je obsegal naslednje teme:

- o razvojni informaciji računalništva in informatike po svetu in v Iskri Delti (A. P. Železnikar)
- trendi v kakovosti programske opreme (R. Piskar)
- sodobna orodja in gradniki za načrtovanje digitalnih vezij (L. Vogel)
- korporativno elektronsko publiciranje (D. Pungerčar)

Drugi pogovor (4. aprila 1989) je obravnaval tele teme:

- razvojni credo Slovenije in Iskre Delte (A. P. Železnikar)
- grafični standardi v luči uporabniške povezave (A. Terčelj)
- grapewine – eksperimentalni distribuirani sistem za elektronsko pošto (Z. Stipetić)
- uporaba VMS mreže v Iskri Delti s poudarkom na elektronski pošti (A. Capuder)
- ekspertni sistemi v Iskri Delti? aktualne novice (A. P. Železnikar)

Pogovore je moderiral prof. dr. Anton P. Železnikar, udeleževali pa se jih uslužbenci delovnih enot razvoja, raziskav, prodaje, vodstva in organizacije.

informacijska kozerija

Na rob zgodbi o računalniškem razumevanju*

Anton P. Železnikar

Ob gozdu, v vinogradu na prisojni strani Alp je stala zidanica, ob njej pa je čemela osamljena, skrivna livada - prostor za zbiranje učenih gozdnih prebivalcev. Nekega dne se je na vratih zidanice pojavil lepak, ki je naznanjal, da bo tekla učena beseda o razumevanju slovenskega jezika. Gozdne živali, ki so ta jezik razumele, se niso mogle načuditi, da je prav nerodni, debelušasti jazbec skonstruiral nekakšen pisani program, ki da njihov jezik razumeva, menda s pomočjo katerega koli delujočega računalnika.

Živali so se čudile same pri sebi, čeprav niso poznale Winogradovega programa SHRDLU, ki naj bi razumeval naravni jezik - seveda angleščino - že v letu 1972. Razpravo o tej sloviti problematiki je po logiki demokratičnih gozdnih navad skliceval vedoželjni suhljati volk, ki je strojno pisarijo nekolikanj osornega jazbeca poskušal razumeti, saj je bila napisana v domačem jeziku. Črke pisarije so bile velike in lahko berljive, čeprav niso bile vselej izvirno slovenske (npr. č, š in ž). Nek zlonamerni dihur je pripomnil, da je jazbec svojo pisarijo zlepil iz tujerodnega, anglosaksonskega črkovja, ki je le nekolikanj drugačno od domačega.

Jazbec je kot temeljit rovokopač dobesedno prekopaval to svojo pisano puščobnost več let. Že njegov oče mu je vcepljal psihološko načelo: čim starejše - tem boljše. Saj kaj pa naj bi delal v preživljanju svojega jazbečjega življenja v tej gozdni osamljenosti ob zidanici, ki je dajala občutje varnosti vsem gozdnim živalim? Napisal je pač to, kar je napisal in zbor gozdnih učenjakov naj bi ta njegov umotvor pregledal in seveda ugodno ocenil, saj je t.i. psihološki čas staranja že potekel.

Tistega dne je vzšlo sonce v jasen dan in živali so veselo hitele po svojih preživetvenih opravkih. Na livadi ob zidanici se je zbrala le najbolj učena in v učenost posvečena elita.

Jazbec se je zavedal svoje moči: o tej stvari je prekopal in prelezel tudi najbolj skrivne rove, v katerih pa ni bilo moč najti kaj posebno užitnega, tj. za strojno razumevanje slovenskega, zlasti mimičnega in kretenjskega jezika bistvenega. Slovnica je bila že utrjeno načelo, o pomenu jezika pa zaradi splošne nejasnosti tako ni bilo vredno zgubljati besed. Zato je razumevanje jezika reduciral na možnost strojnega razumevanja. Vsakomur iz učene druščine na livadi pa je bilo potihoma povsem jasno, da stroj jezika ne more bolje razumeti kot botrica lisica, ki je v razumevanju prava mojstrica in ki zna zase prikrito izkoristiti tudi najmanjšo jezikovno odkritost.

Jazbečev nastop je bil robustno uravnotežen in samozavesten, bil je pravi povzetek napisane učenosti ex cathedra. O kakšni strokovni ali celo bivanjski domačnosti ni moglo biti več govora. Zadeva - računalniško razumevanje jezika - je bilo zrelo psihološko jabolko. Jazbec ga je pač utrgal in použil. Vprašanja niso mogla biti več bistvena. Sicer pa, kdaj neki le je bila učena druščina dovolj učena?

Po jazbečevem nenavadno hitrem, hrupno jeznoritem odhodu je učena druščina staknila glave. Jazbec je prestal preizkušnjo pa naj gozdne živali porečejo, kar hočejo. Jasno je, da mora jazbec kopati še naprej, razširjati svoje rovje in riti v nove smeri razumevanja domačega jezika. Pri tem mu lahko pomaga tudi 2000 mravljincev, ki jih jazbec lahko usmerja v manjše jezikovne vrtine ali celo v mišje rove. Tu ni dileme in življenje domače učenosti se lahko žlahtno pretaka tudi naprej.

V kotu livade, pod velikim listom je ves ta čas dremala žaba. Bila utrujena od nočnega kvakoraja in dogajanju na livadi ni preveč zavzeto sledila. Vendar je nekaj tega, jazbečjega le slišala, veliko njegovega-izrečenega pa tudi prespala. Zato si je lahko oblikovala svoje kritično mnenje, ki je bilo seveda čista žabja modrost. Če je jazbec res zlepil to svojo ne povsem novo pisano učenost iz različnih črk, programov, odstavkov, namigov in sklepov, je vendarle opravil pionirsko delo. Če nič drugega je na gozdnih tleh ponovil napako iz leta 1972, za katero se je pokesal Winograd v letu 1986 nekako takole: "Gadamer, Heidegger, Habermas, and others argue that the goal of reducing even 'literal' meanings to truth conditions is ultimately impossible and inevitably misleading. It focusses attention on those aspects of language (such as the statement of mathematical truth) that are secondary and derivative, while ignoring the central problems of meaning and communication."[1]

---

* Ta zgodba je nastala pred leti nenamerno in popolnoma naključno. Iz tiskarne so sporočili, da je predzadnja stran prazna in da naj se za njeno popolnitev hitro še kaj prinese. Pri iskanju primernega oglasa o kakšni strokovni prireditvi ni bilo sreče. Tako je nastala ta zgodba kot polemična gozdna povest oziroma basen. Kakršnakoli podobnost s fizičnimi ali konkretnimi gozdnimi prebivalci in z livado ob zidanici je lahko zgolj naključna. Zgodba je nastala kot značilna umetnointeligentnostna refleksija avtorja.

[1] "Gadamer, Heidegger, Habermas in drugi trdijo, da je cilj redukcije celo dobesednih pomenov v resničnostne pogoje povsem nemogoč in neizbežno zavajajoč. Ta cilj usmerja pozornost na tiste vidike jezika (take kot je trditev o matematični resnici), ki so sekundarni in izvedljivi, ko prezirajo osrednje probleme pomena in komunikacije."

Kot vsaka zgodba iz sveta živali ima tudi zgodba o razumevanju nekega jezika s strojem svojo moralo. Če koplješ in zbiraš izkušnje, tako kot to delajo Japonci, se bo nekega nepredvidljivega dne vendarle lahko pojavil žepni računalnik, v katerega boš govoril japonsko, njegov zvočnik pa bo oddajal zvočno angleščino. Podobno boš nekaj kasneje lahko vanj govoril slovensko in iz njega bo odmevala zvočna japonščina. Da je ta pot prava, o tem ni mogoče učeno dvomiti. Zamislite si primer, ko bo slovenski nagovor najprej prevajan v zvočno japonščino, iz nje v zvočno angleščino in zopet iz angleščine v zvočno, pomensko nepopačeno, razumljivo slovenščino. Produkt takega tranzitivnega prevajanja iz slovenščine v slovenščino bo kvečjemu t.i. zakasnela slovenščina. O tem okorni jazbec in suhljati volk ne smeta dvomiti - v dobrobit učenega razvoja domače jezikovne vednosti.

DECUS EUROPE AI ANNUAL MEETING

Theme: Towards an Open World

The Digital Equipment Computer Users Society (DECUS) European Artificial Intelligence Working Group will be holding its second annual meeting at the DECUS Europe Symposium. The Symposium will be held in The Hague, Holland from Monday September 18th, 1989 to Friday September 22nd, 1989.

Papers are sought on the following subjects:

* Descriptions of AI Applications
* Tools and Live Applications
* Neural Nets
* Human Computer Interface
* AI Interfaces with Realtime Processes
* AI/DBMS Interfaces
* AI Project Management
* Development Methodologies
* Robotics
* Vision

This is not an exhaustive list. We will be willing to examine any proposition you think might be of interest to other AI developers or users.

A prize will be awarded to the author of the best paper. The Committee will cover the travel and living expenses to the Symposium of the author of the best paper.

For further details please contact:

Guy Delfontaine, Ecole Polytechnique de Lausanne, Dep. Informatique Unité LEAO, 1015 Lausanne, Switzerland.
Tel: +41 21 693 47 66 Fax: +41 21 693 46 60

Attention to the Readers of Informatica

Readers of Informatica are kindly requested to send information on their papers recently published in foreign professional (scientific, also philosophical) journals (periodicals). Such information will be regularly published within this column.

Some Recently Published Papers in Foreign Professional Periodicals

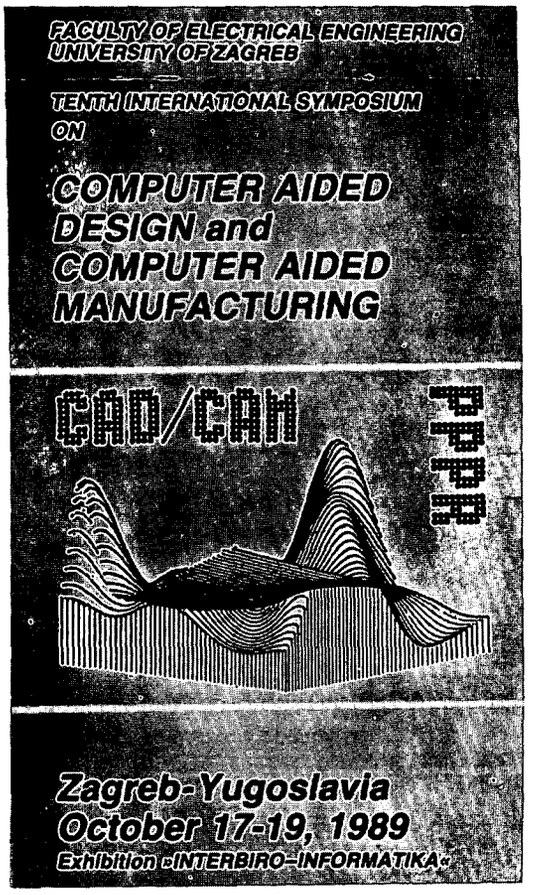
T. Welzer, I. Rozman and J. Gyorkos, Automated Normalization Tool. **Microprocessing and Microprogramming*** *25 (1989) 375 – 380.*

A. P. Železnikar, Information Determinations II. **Cybernetica**** *32 (1989) 5 – 44.*

---

* **Microprocessing and Microprogramming** is published by North – Holland P.C., Amsterdam.

** **Cybernetica** is published by the International Association for Cybernetics, Palais des Expositions, Place Andre Rijckmans, B.5000 Namur, Belgium.



In consideration of the development of new computerized technologies and design techniques, business and manufacturing systems, and current advances in the theory of information, business planning and manufacturnig systems, we have decided to substantially expand and re-group the topics for our jubilee, 10th International CAD/CAM Symposium, which is to be held in Zagreb, **October 17–19, 1989.**

Namely, current developments place an ever increasing emphasis on integration and systems knowledge, with the concurrent abandoning of divisions into specific lines of activity. Therefore, in addition to CAD, CAM, CIM and CAE, the Symposium should also include topics such as expert systems, artificial intelligence and large-scale systems, and review the current state of the art and the envisioned development of these new fields in the near future. These are the reasons why the Programme Committee has substituted new topics, discussing computer tehniques common to different lines of activity, for profession--based sections (e.g. electrical, mechanical, naval and civil engineering, etc).

SYMPOSIUM TOPICS

1. **Methods for the design of deformable bodies**
   (elastic, plastic, viscoelastic, viscoplastic, etc.)
2. **Methods for field design**
   (electromagnetic, thermal, hydromechanical, aeromechanical, optical, etc.)
3. **Information system design**
   (engineering, business, planning, health care, etc.)
4. **Operations and manufacturing processes**
   (methods for the analysis, synthesis, simulation and optimizing of continuous and discrete processes in different engineering systems)
5. **Computer aided design**
   (product design based on a given or assumed basic morphological, topological and geometrical structure derived from primary (functional) and secondary (analytical) processes; final design based on functional, ergonomic, esthetic and other criteria)
6. **New horizonts in computer application**
   (expert system approach, new methods in software engineering, man-machine interface, large scale design, management of application changes in permanent development, impact of new graphics, impact of new languages and coming standards)

PAPER SUBMISSION AND REGISTRATION DEADLINE

The duly filled registration form, with a comprehensively illustrative summary of the paper not exceeding 500 words should be sent to the address of the Technical Organizer – ATLAS, Congress departement, Zagreb – by **November 1, 1988.** The selected papers written in accordance with the instructions which the authors will receive with the notice that their papers have provisionally been accepted on the basis of the summary, should be sent in by **April 3, 1989.** Equipment available to the speakers includes slide and overhead projectors and VHS video large screen projector.

One author may present no more than two papers whether as author or as co-author. Only papers the authors of which will have duly paid their registration fees will be printed in the Proceedings.

TIME AND PLACE

The Symposium will be held at the Congress Centre of the Zagreb Fair, Avenija Borisa Kidriča 2, Zagreb, within the scope of the specialized INTERBIRO-INFORMATIKA exhibition, from **October 17–19, 1989.**

STRUCTURING OF THE SYMPOSIUM

Oral presentations, poster sections, hardware and software presentations, exhibitions.

LANGUAGES

The official languages of the Symposium will be Croato-Serbian and English. Simultaneous interpretation will be provided.

ACCOMMODATION

Accommodation will be provided in A and B category hotels. Detailed information including booking details, will be given in the Second Notice.

ORGANIZING AND PROGRAMME COMMITTEE

President
Dr Zijad Haznadar, Professor at the Faculty of Electrical Engineering, University of Zagreb

Vice-President
Dr Vesna Jurčec, Adviser at the Hydrometeorological Institute of the Socialist Republic of Croatia, Zagreb

Secretaries
Sead Berberović, M.S., Assistant Lecturer at the Faculty of Electrical Engineering, University of Zagreb
Dubravko Pavlović, ATLAS Travel Agency, Congress Department, Zagreb

ADDRESS OF THE TECHNICAL ORGANIZER

ATLAS – Congress Department
Trg senjskih uskoka 7
41020 Zagreb
Telephone: +38/41/525-333, 528-094
Telex: 22413 atlcon yu
Telefax: +38/41/525-468

CALL FOR PAPERS

1990 OSLO-NORWAY

Second European Conference on Software Quality Assurance

May 30 - June 1, 1990, Oslo, Norway

Organized by the EOQC Software Committee

Software Quality Assurance in a Changing Environment

Over the last few years the demand for software quality assurance, and quality software, has increased steadily. New regulations dealing with labeling and certification of software are being issued. At the same time we experience a period of rapid change both in technology and in our business environment. Software that met yesterday's needs does not meet today's requirements. This represents significant challenges to everyone working with software development and quality. In the main part of the conference a variety of topics and application areas will be addressed. Papers related to these are welcome.

The subjects emphasized by EOQC/SQA 90 will be

- **Human factors**
- **Software quality economics and measurements**
- **Contractual issues**
- **Tools and methods for software quality**
- **Organizing for quality software**

We expect papers from the following fields of application

- Large projects - onshore and offshore
- Dataprocessing
- Aerospace/energy
- Banking, finance and insurance
- Consumer products - imbedded software
- Medical applications
- Trans European Projects
- Telecommunications
- Government
- Computer Integrated Manufacturing (CIM)
- Car industry
- Education
- New fields of application

Authors are invited to submit unpublished, original papers, describing the most recent work and experience. Full papers, not exceeding 20 pages, A4, double space typing, written in English, should include the title of contribution, the full name of the authors, affiliation, address, telephone and telefax numbers. Please, send 4 copies of full paper to the conference secretariat. All papers will be reviewed by international experts in the field of software quality assurance. Preliminary selection of speakers will be done, based on abstracts of approximately 1 page. Accepted and presented papers will appear in the Conference Proceedings.

Submission of Abstracts	**June 1, 1989**
Notification of Preliminary Acceptance	**July 15, 1989**
Submission of Papers	**October 1, 1989**
Notification of Papers Acceptance	**December 1, 1989**

Conference secretariat: The Norwegian Computer Society, P.O.box 6714 Rodeløkka, N-0503 Oslo 5 Norway
Telephone +47 2 37 02 13 - Telefax + 47 2 35 46 69

THE NORWEGIAN COMPUTER SOCIETY

EUROPEAN ORGANIZATION FOR QUALITY (SOFTWARE COMMITTEE)

NORWEGIAN SOCIETY FOR QUALITY

Informatica

Informatica *is published four times a year in Winter, Spring, Summer and Autumn by the Slovene Society Informatika, Iskra Delta Computers, Stegne 15C, 61000 Ljubljana, Yugoslavia.*

Volume 13 Number 3 July 1989 YU ISSN 0350-5596

Informatica

A Journal of Computing and Informatics

CONTENTS