88 informatica 3

YU ISSN 0350-5596

Iskra Delta

Iskra Delta
Iskra Delta
Iskra Delta
Iskra Delta
Iskra Delta
Iskra Delta
Iskra Delta
Iskra Delta
Iskra Delta
Iskra Delta
Iskra Delta
Iskra Delta
Iskra Delta
Iskra Delta

proizvodnja računalniških sistemov in inženiring, p.o.

61000 Ljubljana, Parmova 41

telefon: (061) 312-988

telex: 31366 YU DELTA

informatica

JOURNAL OF COMPUTING AND INFORMATICS

YU ISSN 0350-5596

VOLUME 12, 1988 – No. 3

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

Editorial Board

Suad Alagić, Sarajevo; *Damjan Bojadžiev*, Ljubljana; *Jozo Dujmović*, Beograd; *Janez Grad*, Ljubljana, *Bogomir Horvat*, Maribor; *Ljubo Pipan*, Kranj; *Tomo Pisanski*, Ljubljana; *Oliver Popov*, Skopje; *Sašo Prešern*, Ljubljana; *Viljem Rupnik*, Ljubljana; *Branko Souček*, Zagreb

Editor-in-Chief :

Prof. Dr. Anton P. Železnikar

Executive Editor :

Dr. Rudolf Murn

Publishing Council:

T. Banovec, Zavod SR Slovenije za statistiko, Vožarski pot 12, 61000 Ljubljana;

A. Jerman-Blažič, Iskra Telematika, Kardeljeva ploščad 24, 61000 Ljubljana;

B. Klemenčič, Iskra Telematika, 64000 Kranj;

S. Saksida, Institut za sociologijo Univerze Edvarda Kardelja, 61000 Ljubljana;

J. Virant, Fakulteta za elektrotehniko, Tržaška 25, 61000 Ljubljana.

Headquarters:

Informatica, Journal for Computing and Informatics, Iskra Delta Computers, Stegne 15C, 61000 Ljubljana, Yugoslavia
Phone: (+38 61) 57 45 54. Telex: 31366 yu delta
Fax: (+38 61) 32 88 87 and (+38 61) 55 32 61.

Annual Subscription Rate: US$ 30 for companies, and US$ 15 for individuals.

Opinions expressed in the contributions are not necessarily shared by the Editorial Board

Printed by: Tiskarna Kresija, Ljubljana

CONTENTS

informatica

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

Casopis izdaja Slovensko društvo Informatika, 61000 Ljubljana, Parmova 41, Jugoslavija

YU ISSN 0350-5596

LETNIK 12, 1988 – ŠT. 3

Uredniški odbor:

Suad Alagić, Sarajevo; *Damjan Bojadžiev*, Ljubljana; *Jozo Dujmović*, Beograd; *Janez Grad*, Ljubljana, *Bogomir Horvat*, Maribor; *Ljubo Pipan*, Kranj; *Tomo Pisanski*, Ljubljana; *Oliver Popov*, Skopje; *Sašo Prešern*, Ljubljana; *Viljem Rupnik*, Ljubljana; *Branko Souček*, Zagreb

Glavni in odgovorni urednik:

prof. dr. Anton P. Železnikar

Tehnični urednik:

dr. Rudolf Murn

Založniški svet:

T. Banovec, Zavod SR Slovenije za statistiko, Vožarski pot 12, 61000 Ljubljana;

A. Jerman-Blažič, Iskra Telematika, Kardeljeva ploščad 24, 61000 Ljubljana;

B. Klemenčič, Iskra Telematika, 64000 Kranj;

S. Saksida, Institut za sociologijo Univerze Edvarda Kardelja, 61000 Ljubljana;

J. Virant, Fakulteta za elektrotehniko, Tržaška 25, 61000 Ljubljana.

Uredništvo in uprava:

Casopis Informatika, Iskra Delta, Stegne 15C, 61000 Ljubljana, telefon (061) 574 554; teleks 31366 YU Delta; fax (061) 328 887 in (061) 553 261.

Letna naročnina za delovne organizacije znaša 24000 din, za zasebne naročnike 6000 din, za študente 2000 din; posamezna številka 8000 din.

Stevilka žiro računa: 50101-678-51841

Pri financiranju časopisa sodeluje Raziskovalna skupnost Slovenije

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

Tisk: Tiskarna Kresija, Ljubljana

V S E B I N A

UDK 681.3.06 PROLOG:519.682

Monika Kapus-Kolar
IJS, Ljubljana

Osnovna objektna interpretacija jezikov tipa Prolog je kreiranje in brisanje modulov. Nekateri jeziki te družine (npr. Delta Prolog) uvajajo vzporedno/zaporedno kompozicijo in eksplicitno komunikacijo, zaradi česar so primerni za specifikacijo komunikacijskih protokolov. Najvažnejša ideja pričujočega dela je ločitev pojma modula od pojma cilja, tako da lahko vsak modul sodeluje v več ciljih (sestavljenih modulih). Jezik Hornovih stavkov z vzporedno/zaporedno kompozicijo atomičnih dogodkovnih in nedogodkovnih ciljev smo dopolnili z novimi sintaktičnimi elementi, ki v luči nove izvedbene semantike omogočajo bolj jedrnato specifikacijo komunikacijskih protokolov, posebno tistih, pri katerih so potrebne številne operacije na posamezni zapleteni podatkovni strukturi. Jedrnatost dosežemo s temeljito izrabo možnosti, ki jih nudi struktura sestavljenih modulov, ki je ponavadi precej preprostejša kot sintaksa atomičnih modulov.

The basic object paradigm of Prolog-type languages is creation and deletion of modules. Some languages of the family (e.g. Delta Prolog) introduce parallel/sequential composition and explicit communication, what makes them suitable for specification of communication protocols. The main contribution of the paper is separation of the "module" concept from the concept of "goal", so that a module may participate in several goals (compound modules). Some syntactic enhancements have been proposed for the language of Horn clauses with parallel/sequential composition of atomic event and non-event goals, which in the light of the new operational semantics provide for more concise specification of protocols, particularly those requiring multiple operations on a complicated piece of data. The core idea of the new language is full exploitation of the structure of compound modules, which is usually much more simple than the syntax of atomic modules.

1. Introduction

The basic idea behind languages of the Prolog family is to find solutions to a problem (goal) by its reduction into more and more trivial subproblems (subgoals).

A Prolog program, [4], is a set of universally quantified first order axioms (Horn clauses) of the form

$A :- B_1, B_2, \ldots, B_n$

where the **A** and the **B**'s are atomic formulae, also called atomic goals. **A** is called the clause's head and the **B**'s are called its body. The computation proceeds by selection of a goal A_i from the current conjunctive goal $(A_1, A_2, \ldots, A_m)$, which is then reduced with a selected clause

$A' :- B_1, B_2, \ldots, B_k$

where **A** and **A'** must be unifiable via the most common substitution θ. The reduction step transforms the current goal into

$(A_1, \ldots, A_{i-1}, B_1, \ldots, B_k, A_{i+1}, \ldots, A_m)\theta$.

In the process of unification, some of the variables of the initial goal are assigned values, which constitute the output of the computation. The computation terminates successfully, when the initial goal is reduced into an empty goal, but may terminate unsuccessfully, if no further reduction is possible, or not terminate at all. Several successful computations of a program may exist, resulting from various selections of clauses for reduction.

Graphical representation of a Prolog program is an AND/OR tree with AND nodes modelling composition of goals into a clause body and OR nodes enumerating the suitable clauses for reduction of a particular goal. Several goals can be reduced in parallel (AND-parallelism) and several alternative solutions searched for in parallel (OR-parallelism). True concurrency is allowed, if atomicity of reduction steps is preserved.

The inherent AND-parallelism of Prolog programs makes them suitable for operational specification of communication protocols. An AND subtree of the AND/OR tree models a particular execution of a system, while the search of the entire AND/OR tree represents verification of all possible behaviours of the system. Note, that logic programming is also suitable for axiomatic specification and verification of communication protocols, but the paper does not deal with this aspect.

The language has two major deficiencies: First, by introducing AND-parallelism, the execution order is controlled by the goal-subgoal relation only, so it is difficult to describe sequential protocols. Second, communication between goals is implicit and asynchronous via common variables, while partners in communication protocols are usually loosely coupled (not sharing any variables).

To solve the first problem, many Prolog-type languages (e.g. Delta-Prolog (DP) [6,7], Concurrent Prolog (CP) [5], M-Prolog [9]) make distinction between the parallel (¦¦) and the sequential (¦) composition of goals. Declaratively, the two composition operators are equivalent to the AND-operator, but serve to specify the execution order in the spirit of CCS [1], CSP [2] or LOTOS [3].

In the literature, we meet two types of explicit communication: communication on the level of variables and communication on the level of atomic goals. An example of the first type are the "read-only" variables in CP, modelling asynchronous broadcast. An example of the second type are events in DP. Here, the willingness of a module to participate in an event of a particular type is expressed by a goal of a special kind - an event goal, which is successfully reduced in cooperation with some peer event goals of the (other) modules. The DP concept of events allows various cooperation schemes, differing in the number of participating modules, the degree of their synchronization and in side-effects of a particular event, while in CP, a single cooperation scheme is defined. Aiming towards an event-order specification language, the DP concept of events is adopted in the paper and extended.

2. The Architectural Aspect of the Language

An instantaneous representation of a system, described by a set of Horn clauses, is a tree - the **architectural tree**. Nodes of the tree are hierarchical parallel/sequential compositions (trees) of goals. The root represents the top-level structure of the system, i.e. its static architectural components (the initial execution goal). Each atomic goal represents a declaration of a particular module of the system. When an atomic goal is reduced with a clause, the body of the clause is introduced in the tree as a descendant of the goal. A subtree, attached to an atomic goal, represents dynamic architecture of the module, declared by the goal. After a node has been successfully executed (all its atomic goals reduced to TRUE), it is deleted from the tree.

As the overall activity of a system is represented as creation and deletion of modules, there is no evident distinction between a module, representing a quasi-static architectural component of the system, and a module, representing a procedure. Such semantic distinction belongs to a lower level of abstraction. The only feature that matters is the capability of the language to specify loosely coupled modules. Modules are declared "loosely coupled" simply by keeping their variable-sets disjoint.

Atomic goals, which are event goals, do not generate subtrees, but are reduced in events. Looking at an event as a common action of several modules, it is a free module, not embedded in the architectural tree, with its submodules residing in various nodes of the tree. According to the previous paragraph, it is difficult to say which module does a particular event goal belong to, but it is no doubt that it belongs to a certain node. As events should serve for cooperation between loosely coupled modules, it is reasonable to declare that event goals, participating in a particular event, must not belong to the same node.

The word "goal" should not be used in the architectural context - it denotes the intention of the current architecture to change, but at that point, we are only interested in an instantaneous picture of a system. Therefore, atomic goals are rather called **atomic modules**. They are basic elements of nodes and are usually combined into **compound modules** by parallel/sequential composition operators.

Modules are further classified into **explicit** and **implicit** ones. They can be best identified by observing a clause body (e.g. Fig.1):

---

compound module: (A¦B¦(C¦¦D¦¦(E¦F)))

explicit modules:

A, B, C, D, E, F
(E¦F)
(C¦¦D¦¦(E¦F))
(A¦B¦(C¦¦D¦¦(E¦F)))

implicit modules:

(C¦¦D), (D¦¦(E¦F)), (C¦¦(E¦F))
(A¦B), (B¦(C¦¦D¦¦(E¦F)))

Fig.1: Explicit and implicit modules of a compound module.

Atomic modules and bodies are explicit modules. Operands of parallel and sequential composition operators are explicit modules. Any proper subset of modules with more than one element, belonging to a parallel composition of modules, is an implicit module and itself a parallel composition of modules. Any proper subsequence of modules with more than one element, belonging to a sequential composition of modules, is an implicit module and itself a sequential composition of modules.

The above definition illustrates the **module-grouping and module-ordering role of** the two **composition operators**. The parallel composition operator groups modules into sets and the sequential composition operator groups modules into sequences. Explicit modules are the actual and implicit modules the potential groups of modules.

3. The Operational Aspects of the Language

3.1. Compound Modules as Goals

If a module is declared to be a **goal**, it specifies a pending action. The goal becomes "executed", when the action is no longer pending. A node gains the right to be deleted from the architectural tree, after all its goals have been executed.

In DP, only atomic modules are goals, every atomic module is a goal and the pending action of each goal is its reduction into TRUE. The main contribution of the paper is the idea, that **the concept of module should be strictly separated from the concept of goal, so that a module might participate in several goals (compound modules)**. Note the importance of the word "participate" - a module itself is not

necessary a goal, but every module should be included into some goal, otherwise it has no practical role in the system.

Motivation for the new idea has been the fact that in most cases events serve just for some kind of unification of modules, belonging to various nodes. From this aspect, event goals are just communicated pieces of data. Assuming that there is a group (composition) of modules, there might be several nodes, each interested in a particular subgroup of the group and willing to observe the whole subgroup in a single event. It is much more elegant to enumerate members of the group once and to declare that each subgroup of the group is an event goal, than to enumerate all subgroups as atomic event goals. If a group of modules is a set (parallel composition), it might serve as a data-base (see section 5 for the example in Fig.6); if it is a sequence (sequential composition), it might serve for specification of a data-stream with multiple observers, each waiting for a particular subsequence. If the unifying event goals are parallel compositions of modules, the unification rule could be less strict (unification possibly preceded by permutation of modules) than for event goals, which are sequences. The fact is, that sequences and sets already exist in clause bodies, so why not to use them (section 5). Similarly, if there has been a non-event goal (a procedure) executed, why should its declaration part be duplicated just to communicate its exit results to an observer.

The central operational concepts of our language are **reduction goals** and **event goals**. The two names indicate that the concept of reduction has been separated from the concept of event - without this step the realization of our ideas would not be possible. The role of reduction goals is twofold: First, if they are atomic, they enforce execution of procedures in the ordinary sense (like non-event goals in DP). Second, they control execution order by imposing hierachical execution of goals: No goal can be executed, if some of its submodules are pending reduction goals, so that no event can occur on exit results of a procedure, until they are available. Because of the nature of the two roles, only explicit modules may be reduction goals, so that a reduction goal must be entirely contained in another reduction goal, or not at all.

Event goals support the idea of overlapping goals, as they may be explicit or implicit modules and may partially overlap. An event goal is executed by an event, where the participating part of a node is exactly the module, representing the goal. Event goals impose no particular execution order - a dangerous dimension of freedom, which is partially compensated with the execution-ordering role of reduction goals and composition operators.

Depending on its reduction type (see section 3.3.), execution of a reduction goal is independent from events or a reduction goal is a submodule of an event goal and executed simultaneously with the event goal.

In DP, event goals never generate descendants in the architectural tree, while event goals with atomic submodules, which are reduction goals, requiring the ordinary type of reduction, do. Nevertheless, when such an event goal gains permission for execution, all such reduction goals within it have already been executed and their descendants deleted from the tree.

3.2. The Execution-ordering and the Selection Role of Composition Operators

The basic role of composition operators is their module-grouping and module-ordering role, but for easy specification of sequential protocols they must also be assigned the **execution-ordering role**. To employ the full power of the two roles, it must be possible to use them independently, while in DP they are not separated. Therefore we define that the parallel and the sequential composition operator have no a priori execution-ordering role.

When considering the execution-ordering role of composition operators, the distinction between compound modules, which are sets, and those, which are sequences, is irrelevant, because the attribute parallel/sequential, attached to composition operators, applies to another role. Therefore, all compound modules will be treated as sequences of modules.

A composition operator separates a sequence of modules into the left and the right subsequence. A DP sequential composition operator forces the goals from the left subsequence to be executed before the goals from the right subsequence. But if a goal extends to the left and to the right subsequence, which subsequence does it belong to? Another question: Should the composition operator delay actions on the right subsequence until the reduction goals from the left subsequence have been executed or until the event goals have been executed or, perhaps, just until all goals from the left subsequence have been created. The answer depends on the nature of a particular application, but as default we propose that creation of all goals of a node is an atomic action and that execution of a goal is treated as atomic in the sense that if a goal A must be executed before a goal B, then (at least virtually) the execution of the goal B may not even start before the execution of the goal B has been completed.

The execution order is effected by reduction goals, implying that some goals must be executed before some others. Beside that, some reduction goals are executed simultaneously with an event goal, implying an OR composition of relations, specifying simultaneous execution. In addition, the language should facilitate specification of further relations of the "before" type.

Such a specification should be concise, therefore we propose that goals are referred to simply by their position in the node and that the composition operator, to which a particular "before" relation is attached, is carefully selected such that the goals of the relation are easily specified relatively to the position of the operator (section 5).

The last requirement suggests distribution of such relations all over the node. Anyway, the execution order is determined by considering all the relations simultaneously (in DP, it is sufficient to consider relations, implied by sequential composition operators, in a particular order). If a set of relations is in contradiction (i.e. $(a<b)$ and $(b<a)$), they are ignored. With that rule, a programmer is free to specify any relation, and if possible, it will be respected (e.g. Fig.2).

In comparison to LOTOS, the language lacks two important features: a construct for expressing intelligent selection (guarded commands) and a construct for expressing disruption of processes. If guards are not located at the meta level (as, for example, in Two-level Prolog [8]), but at the object level, both problems

have a simple common solution - **exclusive composition operators**.

```
body: (a|(ULS<RS)
       b|(RS.A<URS.ULS)
       c|(LS.A<URS)
       d)#M: (event goal = true;
              reduction state = no-reduction)#

execution-ordering relations:

1.composition operator:
  ((a), (a|b), (a|b|c), (a|b|c|d)) <
  ((b), (c), (d), (b|c), (c|d), (b|c|d))

2.composition operator:
  ((c), (d)) < ((b|c), (a|b|c), (b|c|d),
                (a|b|c|d))

3.composition operator:
  ((a), (b), (c)) < ((d), (c|d), (b|c|d),
                     (a|b|c|d))

ignored relations:
  ((b)<>(a|b|c|d),
   (c)<>(a|b|c), (c)<>(a|b|c|d),
   (d)<>(a|b|c), (d)<>(a|b|c|d))
```

Fig.2: The execution-ordering role of composition operators.

A compound module may be a non-exclusive or an exclusive composition of modules. The non-exclusive forms of the sequential and the parallel composition operator are ; and ||, while their exclusive forms are / and //, respectively. Each module of an exclusive composition of modules is attached a set of goals (drawn from the set of all goals of the node) representing the guard of the module (see section 5 and Fig.3 for an example).

```
body: (((a#M: (reduction state = pending;
               reduction type = weak-common)#
         |b#M: event goal = true#
         )
        |c
        |d#M: event goal = true#
       )[(11)M]
      //
       (e#M: (reduction state = pending;
              reduction type = weak-common)#
       |f
       |g
       |h#M: event goal = true#
       )
      //
       (i#M: (reduction state = pending;
              reduction type = weak-common)#
       |j#M: event goal = true#
       )
      )#M: reduction state = no-reduction;
        WM: event goal = true#

scenario:
  execution of event goal
     (((a|b)|c|d)//(e|f|g|h)//(i|j))
  -> ((a|b)|c|d) not ready for selection,
     (e|f|g|h) and (i|j) ready for selection
  -> the node transformed into (e|f|g|h)
     or into (i|j)
  -> execution of event goal (h)
     (or event goal (j))
```

Fig.3: An example of a guarded command.

After the guard of a module has been executed, the next operation on the module may only be its selection for further execution or its deletion from the system. Several modules with executed guards may exist, but exactly one of them is selected non-deterministically and the entire exclusive composition of modules replaced by the selected module, which from that moment behaves like an ordinary module. Because guards are regular processes of a system and one of the modules in exclusive composition disrupts the others, this is a model of process disruption, more general than the one of LOTOS.

3.3. Execution of Reduction Goals

In DP, a reduction goal is executed either by the ordinary type of reduction (reduction of a goal into more and more trivial subgoals), or by participation in an event as exactly the whole event goal. In our language, a reduction goal can also be executed in event, where it is just a submodule of the relevant event goal. Hence, according to this criterion, there are three types of reduction.

1. **On-the-spot reduction** is the ordinary type of reducfion and could serve for specifying an internal process of a node. It may be **non-trivial** or **trivial**. Reduction of an atomic module is non-trivial, as it potentially creates descendants in the architectural tree. Reduction of a compound module is trivial, as it is just an observation that all the reduction goals within the module have been executed.

2. **Strong common reduction** takes place, when a module is a reduction goal and an event goal at the same time (the DP-type event goal). When the event goal is executed, the reduction goal is executed, too. The adjective "common" steams from the fact that the participants can execute the reduction as a common action, without any of them executing the reduction on the spot. Common reduction could serve for exchange of values of any origin.

3. **Weak common reduction** is like strong common reduction, but the reduction goal may also be just a submodule of the relevant event goal.

To facilitate observation of final (exit) results of modules, we introduce a fourth type of reduction, which is a weak common reduction with some additional requirements:

4. **Observed reduction** takes place, when a reduction goal participates in an event, in which one of the participating event goals contains a submodule, unifiable with the reduction goal. That submodule must be an executed reduction goal with reduction type "on-the-spot" or must have been unified with such a goal in one of the previous events. In this way, the node does not have to execute the reduction goal on the spot (that might be a difficult operation, if the event goal is atomic), but just gathers the necessary results by observing someone, who has already executed a matching goal on the spot or has learned the results from another node. Observation of exit results is important from several aspect: First, it can strongly reduce the execution effort. Second, it can reduce non-determinism by forcing various nodes to accept the same solution to various incarnations of the same problem. Third, it could serve to implement monitors of overall system activity. Fourth, the concept of exit results is another step towards LOTOS.

On-the-spot reduction may be thought of as being executed in the node, containing the

reduction goal, observed reduction as executed in another node and common reduction as executed in the space between nodes. Observed and common reduction are treated as trivial, as they do not create descendants.

At the time of its creation, each module is assigned its reduction type and state. The possible reduction types are the four types, declared above. The possible states are "executed", "pending" and "no-reduction".

If a module is a reduction goal, its initial reduction state is "pending" and its initial reduction type may be any type. If a module is not a reduction goal, its initial reduction state is "no-reduction" and its reduction type is "weak-common". When a reduction goal is executed, its reduction state is set to "executed".

If reduction type of a reduction goal is "strong-common", the module is an event goal by definition. If reduction type of a reduction goal is "observed" or "weak-common" and the module is not a submodule of any event goal, it is an event goal by definition.

Implicit modules are never able to propagate exit results of an on-the-spot reduction, while explicit modules are always able to do that. When created, reduction goals with the reduction type "on-the-spot" are assigned unique reduction identifiers. Reduction identifiers of other modules are undefined, until they begin to propagate results of an on-the-spot reduction. In that case, their reduction type and state are set to "on-the-spot" and "executed" and their reduction identifier is set to the identifier of the reduction they propagate. Reduction identifiers have been introduced to distinguish among various incarnations of a procedure and to facilitate specification of events with a limited number of active roles, in which the participating event goals may enroll (see section 5 for the example in Fig.7).

Each atomic reduction goal with reduction type "on-the-spot" is attached a predefined or user-written procedure for unification with clauses' heads. Similarly, heads are attached procedures for unification with reduction goals. Such procedures may be treated as sets of constraints. A necessary condition for reduction of a reduction goal with a particular clause is, that their proposed constraints can be satisfied simultaneously. The reduction is executed exactly according to the constraints, proposed by the goal or the head. The default unification procedure is the most common substitution.

3.4. Execution of Event Goals

An event goal can execute, if it meets a group of suitable event goals. Execution must be fair in the sense that an event goal, which is ready to execute, must not wait indefinitely, if suitable groups keep occurring, and must be allowed to join a group, which is able to accept it. If there is more than one suitable group, the event goal selects between them non-deterministically. While an event goal is waiting for execution, its variables might be getting assigned by other goals, so its selectivity improves with time.

Each event goal is attached a predefined or user-written procedure for its execution. A necessary an sufficient condition for a group of event goals to realize an event is, that their proposed constraints can be satisfied simultaneously. The event is executed exactly according to the constraints, proposed by at least one participant.

When a group of event goals is ready to execute a common event, the pending event is created in the system as a free module, which may (or may not) later be executed (deleted from the system). Each of the participating event goals (according to its synchronization requirements) may be executed simultaneously with the event or before the event, but never after the event. Event goals remain formal participants of an event, until the event has been executed, even if they have already been executed or even deleted from the system. If not forbidden by the existing participants, new participants (sometimes even an unlimited number of them) may join with time. In that case, a participant, which has not requested an immediate execution, can not be executed while new participants could potentially change its execution results (e.g. if they could assign some of its remaining variables). Regardless the type of the execution procedure, an event must mandatory execute all reduction goals, that are submodules of the participating event goals, and must not introduce any new tight coupling of nodes.

We propose to classify requirements, posed by execution procedures, into those considering

1. the number of participants,

2. timing relations (e.g. synchronization, delays),

3. relation between the state of the participants before and after the event, and

4. potential side-effects.

The proposed defaults are: any non-zero number of participants, full synchronization (all event goals executed simultaneously with the event), no special side-effects and the third relation defined by the following unification procedure:

The procedure first tries to make the participating event goals unifiable via substitution, together with a legal distribution of roles. If it succeeds, the event goals are unified via substitution and their pending reductions executed. The exclusive and the non-exclusive version of composition operators are treated as equivalent.

To make the event goals unifiable via substitution, the procedure may apply to them the following transformations, which are not visible after the event:

- permutation of modules in parallel composition,

- grouping of modules (creation of explicit modules for the time of the execution of the event). The new modules are neither reduction nor event goals and are unable to propagate results of an on-the-spot reduction.

The first transformation type supports the idea of using parallel and sequential composition operators for data ordering and reflects the fact that modules in parallel composition are not ordered. The second transformation type facilitates structured observation of unstructured data.

Usually, an event goal (EG) can be transformed in several ways to meet the requirements. In that case, the selection between the transformations is non-deterministic and the procedure

acts as a generator of permutations of modules (e.g. Fig.4).

1.EG: (A¦¦B) - ready to receive data into A and B
2.EG: (a¦¦b) - ready to send data-items a and b

possible transformations before unification and the resulting EG:

	1.EG	2.EG	1.EG after unification A B
1.	(A¦¦B)	(a¦¦b)	(a¦¦b)
2.	(A¦¦B)	(b¦¦a)	(b¦¦a)
3.	(B¦¦A)	(a¦¦b)	(b¦¦a)
4.	(B¦¦A)	(b¦¦a)	(a¦¦b)

Fig.4: An event, which does not preserve the order of data.

When the event goals have been made unifiable via substitution, sets of corresponding modules can be identified (e.g. Fig.5). We are interested only in the explicit modules of the event goals (the event goals themselves are treated as explicit modules).

1.EG: (a¦(b¦¦C¦¦d))
2.EG: (a¦(X¦¦c¦¦d))

sets of corresponding modules:

{(a),(a)); {(b),(X)}; {(C),(c)); {(d),(d)}
{(b¦¦C¦¦d),(X¦¦C¦¦d)};
{(a¦(b¦¦C¦¦d)),(a¦(X¦¦c¦¦d))}

Fig.5: An example of sets of corresponding modules.

A distribution of roles between the participating event goals is legal if the distribution of reduction types, states and identifiers of the corresponding modules is legal for all sets of corresponding modules.

We define the procedure for checking the distribution for a particular set of corresponding modules from the point of one of the modules of the set. The distribution from the aspect of each module of the set must comply with the following rules:

1. If the reduction type of a module is "no-reduction", any distribution is legal from the point of the module.

2. If the reduction type of a module is "on-the-spot", the corresponding modules must not have this property, unless they have the same reduction identifier.

3. If the reduction state of a module is "pending", the required reduction must be trivial.

4. If the reduction type of a module is "observed", there must exist a corresponding module with the reduction type "on-the-spot".

5. If the reduction type of a module is "strong-common" and its reduction state is "pending", the module must be a whole participating event goal.

If the distribution of roles is legal, the event goals are unified by a procedure, which differs from the default procedure for unifying atomic goals with clause heads only in the way of unifying variables, which remain unassigned in the event: The corresponding variables are not unified into a single variable, but retain their identities to prevent tight coupling of loosely coupled modules.

The modules of the event goals, which have a corresponding module with the reduction type "on-the-spot" and are able to propagate results of an "on-the-spot" reduction, are assigned this property.

4. The Problem of Verification

The ease of verification of specifications in the new language depends on the nature of procedures for execution of goals. For the case of the proposed default procedures we provide some basic guidelines, how clauses of the language could be converted into the familiar and widely treated form with just atomic event and non-event goals, the classical parallel and sequential composition, guarded sequences of goals and synchronous events, which require just pure unification.

The behaviour of a node can be represented as a set of possible execution sequences of "on-the-spot" reduction goals and event goals. If a reduction goal is executed simultaneously with an event goal, it is not represented separately. As goals are not executed more than once, the set is finite and so are the sequences. Hence, they can be specified with the classical composition operators and guards. Each event goal is then replaced by an OR composition of all its forms, that can be generated by grouping and permutation of submodules. Changing any term into an atomic goal is just a syntactic operation, but an exotic component of the language still remains, namely the procedure for execution of event goals.

The problematic part of the proposed default execution procedure is the role distribution checking part. It involves checking and setting of the three implicit variables, associated with each submodule of the participating event goals: the reduction type, state and identifier. When changing an event goal into an atomic goal, its modular structure must be retained, so that every submodule can explicitly be attached the three reduction variables. As the values of the variables are passed between goals, every variable must have its input and its output copy. Analogously, each atomic non-event goal must be attached a variable for generation of its reduction identifier.

5. A Simple Form of the Language and Some Examples

An elegant and detailed syntax for the language is beyond the scope of the paper. To be able to provide some examples, just a very simple form of the language is proposed informally.

A specification is a set of Horn clauses with the following conventions:

a) All composition operators (¦, ¦¦, /, //) should be used as infix operators. If all types of composition operators are treated as one, a clause's body is a hierarchical sequence of modules. For each composition operator it is obvious, which is the sequence, it helps to create. Each module of a sequence is explicitly

or implicitly followed by its belonging composition operator. The presence of a composition operator may be implicit, if it belongs to the last module of a sequence and no comment needs to be attached to it. If there is a comment, attached to a module, it may (from the aspect of the syntax) also be treated as attached to the belonging composition operator.

In a comment, attached to a composition operator, various subsequences of the body can be defined, relatively to the position of the operator, with the following syntax:

B: the body.

M: the module, belonging to the composition operator.

S: the sequence, created by the composition operator.

S(n); n:(0),1,2..: S(0) = S. S(n+1) is the sequence, to which S(n) belongs as the rightmost element of its left subsequence.

LS(n): the left subsequence of the sequence S(n). The left subsequence of S is the subsequence to the left of the composition operator.

RS(n): the right subsequence of the sequence S(n). The right subsequence of S is the subsequence to the right of the composition operator.

(nm)X: the subsequence of a sequence X, starting with the n-th element and ending with the m-th element of the sequence X. The constant s denotes the size of a sequence X. Note: If X is a sequence with a single element, (11)X denotes the first element of the element, not the element, and (mn)X denotes a subsequence of the element. Sequences with a single element, which is again a sequence with a single element, are forbidden.

b) Sets of modules, referred to in comments, may be constructed by intersection (.), union (+) and difference (\) from the following simple sets:

X: the set of all modules, belonging entirely to a sequence X.

WX: the module, which covers exactly the whole sequence X.

UX: the set of all modules, containing at least a part of a sequence X.

E: the set of all explicit modules of the body.

I: the set of all implicit modules of the body.

A: the set of all atomic modules of the body.

C: the set of all compound modules of the body.

G: the set of all modules of the body, which are goals.

RG: the set of all modules of the body, which are reduction goals.

EG: the set of all modules of the body, which are event goals.

c) Each explicit module may be followed by a comment #...#, declaring the non-default properties of sets of its submodules and itself. If a property of a particular module is defined in a comment, attached to an explicit module, and again in a comment, attached to one of its explicit submodules, the first declaration is ignored. All specified sets of modules are implicitly in intersection with M.

The non-predefined properties of modules are: to be a reduction goal, to be an event goal, the reduction type and, if the module is an event goal, the required number of participants of the event, in which it will be executed.

d) Each composition operator may be followed by a comment {..}, enumerating some execution-ordering relations in the form S1<S2 (the goals of the set S1 must be executed before the goals of the set S2). All specified sets of modules are implicitly in intersection with G.

e) Each module in an exclusive composition may be followed by a comment [..], specifying its guard. All specified sets of modules are implicitly in intersection with G.

f) The defaults are those, proposed earlier in the paper, plus:

- Modules are not event goals.
- Atomic modules are reduction goals with reduction type "on-the-spot".
- Compound modules are not reduction goals.
- Execution-ordering relation of ; and / operators: {A.RG.LS<URS}.
- Execution-ordering relation of ;; and // operators: {}.
- Guards: [A.RG.(11)M], if the module is compound, or [RG.M], if the module is atomic.

```
system:- sender;;receiver1;;receiver2.

sender:- ((wait1;;wait2)#A: (event goal = true;
                             reduction state =
                               no-reduction;
                             participants = 2)#
         ;{EG.LS<EG.RS}
          (a(X);;b(Y);;c(Z)
          )#M: event goal = true#
         ).
a(X):-.........
b(X):-.........
c(X):-.........

receiver1:- ((c(X);;a(Y)
              )#WM: event goal = true;
                A: reduction type = observed#
             ;()
              wait1#M: (event-goal = true;
                        reduction state =
                          no-reduction;
                        participants = 2)#
             ;{LS<RS}
              process1(X,Y)
             ).
process1(X,Y):-.........

receiver2:- ((a(X);;b(Y)
              )#WM: (reduction type =
                       strong-common;
                     reduction state = pending);
                A: reduction type = observed#
             ;()
              wait2#M: (event goal = true;
                        reduction state =
                          no-reduction;
                        participants = 2)#
             ;{LS<RS}
              process2(X,Y,Z)
             ).
process2(X,Y,Z):-.........
```

Fig.6: Distribution of subsets.

With these defaults, protocols can be specified entirely in the classical style and (with a slightly modified execution procedure for

events) the language used as a dialect for the event-ordering part of LOTOS. Next, we provide some motivation examples for the new features of the language.

Example in Fig.6: A set of data-items is sent to a community of modules, so that every module receives exactly the items of its personal interest in a single event. The sender produces data and waits for creation of the receivers. Whenever all the data-items, interesting for a particular receiver, are generated, they are transmitted and, because of the fairness requirements, the receiver actually receives them.

```
system:- active1||active2||passive.

a(X):-.........
b(X):-.........

active1:- (( a(X)#M: reduction type = observed#
             ||a(Y)
             ||b(Z)
            )#WM: event goal = true#
           |(LS<RS)
            process1
           ).
process1:-.........

active2:- (( a(X)
             ||a(Y)#M: reduction type = observed#
             ||b(Z)#M: reduction type = observed#
            )#WM: event goal = true#
           |(LS<RS)
            process2
           ).
process2:-.........

passive:- (( a(X)
             ||a(Y)
             ||b(Z)
            )#WM: event goal = true;
              A: reduction type = observed#
           |(LS<RS)
            process3
           ).
process3:-.........
```

Fig.7: Distribution of data from several sources and the concept of roles.

Example in Fig.7: A synchronous event is specified, involving collection of data-items from several sources, merging of data into a compound message and its dissemination to all modules of the system. **active1** and **active2** will wait for each other to execute the first event, but will not wait for the passive observer, if its event goal is created to late. If the event was asynchronous, allowing an unlimited number of participants, the passive observer could receive the message regardless of the execution speeds. There are three roles in the event (a, a and b), the first played by **active2** and the others by **active1**.

```
sender:- (Address#M: event goal = true#
         |(LS<URS)
          message
         )#WM: event goal = true;
           A: reduction state = no-reduction#.
```

Fig.8: Receving an address and sending a message to the address.

The task in example from Fig.8 is to receive the address, on which a particular message is to be sent, and to send the message.

6. Conclusions

The basic object paradigm of Prolog-type languages is creation and deletion of modules. Some languages of the family (e.g. DP) introduce parallel/sequential composition and explicit communication, what makes them suitable for specification of communication protocols.

The main contribution of the paper is separation of the "module" concept from the concept of "goal", so that a module may participate in several goals (compound modules). Some syntactic enhancements have been proposed for the language of Horn clauses with parallel/sequential composition of atomic event and non-event goals, which in the light of the new operational semantics provide for more concise specification of protocols, particularly those requiring multiple operations on a complicated piece of data. The core idea of the new language is full exploitation of the structure of compound modules, which is usually much more simple than the syntax of atomic modules.

Three independent roles of composition operators have been identified: In the module-grouping and module-ordering role they facilitate specification of sets, subsets, sequences and subsequences of modules. In the execution-ordering role they facilitate specification of the execution order of pending goals. In the selection role they facilitate specification of guarded commands and process disruption.

The concept of reduction has been separated from the concept of event by introducing independent promotion of modules into reduction goals or into event goals. Reduction goals have as well been assigned the execution-ordering role.

To indicate to which extent the execution of a reduction goal depends on execution of an event goal, four reduction types have been introduced: on-the-spot reduction, strong common reduction, weak common reduction and observed reduction. The most original one is the observed reduction, which could serve for specifying observation of exit results, for specifying events with a limited number of roles, for reduction of the computational effort, for reduction of non-determinism and for implementation of monitors of the overall system activity.

The paper indicates, how the architectural and the behavioural aspect of a system can be unified into a single semantic model and efficiently specified by a simple language.

References

[1] R.Milner: A Calculus of Communicating Systems, Springer verlag, LNCS 92, Berlin 1980

[2] C.A.R.Hoare: Communicating Sequential Processes, Communications of the ACM, vol.21, no.8, 1978, pp.666-677

[3] E.Brinksma: A Tutorial on LOTOS, in "Protocol Specification, Testing, and Verification, V", M.Diaz ed., North-Holland 1986, pp. 171-194

[4] L.M.Pereira, F.C.N.Pereira, D.L.D. Warren: User's Guide to DECsystem-10 PROLOG, Occasional Paper 15, Dept.of AI, Edinburg 1979

[5] E.Y.Shapiro: A Subset of Concurrent Prolog and Its Interpreter, ICOT TR-003 (1983)

[6] L.Monteiro: A Proposal for Distributed Programming in Logic, in "Implementations of Prolog", J.Campbell ed., Ellis Horwood 1984

[7] L.M.Pereira, R.Nasr: Delta-Prolog: A Distributed Logic Programming Language, Departamento de Informatica, Universidade Nova de Lisboa, 1985

[8] A.Porto: Two-level Prolog Departamento de Informatica, Universidade Nova de Lisboa, 1985

[9] M-Prolog, Language Reference Manual, Institute for Co-ordination of Computer Techniques, Budapest, March 1983

ISMM

CALL FOR PAPERS

ISMM International Conference

MINI AND MICROCOMPUTERS

From Micros to Supercomputers

December 14-16, 1988
Miami Beach, Florida

SPONSORED BY
The International Society for Mini and Microcomputers (ISMM)
Technical Committee on Computers

SUPPORTED BY
Department of Electrical and Computer Engineering, University of Miami, Coral Gables, Florida, U.S.A.
Modcomp, an AEG Company, Fort Lauderdale, Florida, U.S.A.

LOCATION
Hotel Hilton Fontainbleau, Miami Beach, Florida, U.S.A.

SCOPE
Covers all aspects of computer architecture, organization, and design, artificial intelligence, software systems, and computer applications.

- Computer architecture
- Parallel computers
- Fault-tolerant systems
- VLSI and chip design
- RISC architecture
- Expert systems
- Computer applications
- Parallel and concurrent programming
- Operating systems and databases
- Software engineering
- Real-time systems
- Artificial intelligence
- Supercomputing
- Computer vision
- Robotics
- AI programming and environment
- Software systems
- Local area networks
- Optimizing compilers

Three tutorials will be presented by leading experts in the field. Tentative topics are: Advanced Computer Architecture, Parallel Computers and Artificial Intelligence.

SUBMISSION OF PAPERS

Five copies of a 400-word summary	June 15, 1988
Notification of authors	July 15, 1988
Full papers in camera-ready form	October 1, 1988
Conference	December 14 - 16, 1988

All papers shall be reviewed for possible publication in one of the ISMM journals: International Journal of Mini and Microcomputers, and Microcomputer Applications.

INTERNATIONAL PROGRAM COMMITTEE

G. Rabbat, General Chairman	U.S.A.	M. Kabuka	U.S.A.	V. Oklobdzija	U.S.A
B. Furht, Program Chairman	U.S.A.	P. Liu	U.S.A.	D. Petkovic	U.S.A.
P. Alpar	U.S.A.	E. Luque	Spain	A.M. Salem	Egypt
M. Carapic	U.S.A.	N. Marovac	U.S.A.	B. Soucek	U.S.A.
R. Bisiani	U.S.A.	G. Mastronardi	Italy	W.V. Subbararo	U.S.A.
E. Fernandez	U.S.A.	J.D. Meng	U.S.A.	D. Tabak	U.S.A.
D. Gulch	U.S.A	V. Milutinovic	U.S.A.	M. Tapia	U.S.A.
M. Hamza	Canada	D. Moldovan	U.S.A.	J. Urban	U.S.A.
C.C. Hsu	R.O.C.	S.C. Moon	Korea	F. Vajda	Hungary
T. Ishiko	Japan	B.N. Naumov	U.S.S.R.	P. Visuri	Finland
				M. Vuskovic	U.S.A.

ADDRESS
For correspondence, submission of extended summaries, and to be placed on the mailing list, write to: Dr. B. Furht, Director of Advanced Technology, Modcomp, 1650 West McNab Road, P.O. Box 6099, Mail Stop 850, Ft. Lauderdale, FL 33340-6099, U.S.A.

Please complete and return this form to: ISMM Secretariat, P.O. Box 25, Calgary, Alberta, Canada T3A 2G1

Please send me information concerning:
- ☐ The International Journal of Mini and Microcomputers
- ☐ Microcomputer Applications Journal
- ☐ The International Journal of Robotics and Automation
- ☐ Expert Systems, Los Angeles, December 1988
- ☐ Computer Applications in Design Simulation and Analysis, Reno, Nevada, U.S.A., February 1988
- ☐ Reliability and Quality Control, Los Angeles, December 1988.
- ☐ Send me call for papers to conferences in the following areas: _____

Tatjana Welzer
Ivan Rozman
József Györkös
University of Maribor

UDK 681.3.06 PROLOG:519.24

After a brief presentation of base and some definitions neded, the aim of the article is to deal with the normalization of relational database.The implementation of normalization process will be shown by PROLOG, the fifth generation language. The ascendancy of this language over the languages of the fourth generation is that the program is not defined as a sequence of steps but as a discription of relations between objects. This enables a logical conclusion in questions that are put in various ways. The built-in database of PROLOG will be exploited as well.

Po kratki predstavitvi relacijske baze in definicij, ki jih potrebujemo v nadaljevanju, bomo v članku predstavili proces normalizacije relacijske baze podatkov. Izvedbo procesa normalizacije bomo predstavili s PROLOG-om, jezikom pete generacije. Prednost tega jezika pred jeziki četrte generacije je v tem, da program ne definiramo kot zaporedje korakov, temveč kot opis relacij med objekti, kar omogoča logično sklepanje ob različno zastavljenih vprašanjih. Prav tako pa bomo izkoristili tudi PROLOG-ovo vgrajeno bazo podatkov.

1.Ø INTRODUCTION

One of the most important modules for various computer applications is database. The database can be a hierarchical database, a network database or a relational database. The latter is more and more often used instead of the first two databases and is successfully gaining ground on all fields off application. Its advantage is above all in its simple and user friendly transfer from the paper to the computer. Successful work with the relational database can be ensured only by a thoughtful formulation of base. Because of its extensiveness this formulation should be aided by a corresponding tool.

2.Ø RELATIONAL DATABASE

Relational database is a temporally variable multitude of relations /ALAG84/. R(A1,A2,....,An) is the relation over multitudes D1,D2,...,Dn if the submultitude of Descartes product is as follows:

$$R \subseteq \{ D1 \times D2 \times \ldots \times Dn \}$$

D1,D2,....,Dn mean the domains of attributes. The attributes A1,A2,...,An denote the structure of the relation which is called **relational schema.**

2.1.Ø RELATIONAL ALGEBRA

Relational algebra is a simple formal language which enables data manipulation /CODD7Ø/. Relational algebra consists of operations over multitudes (union, intersection and difference) and special operation (projection,selection,join and division). For our further work, operation of projection is of main importance.

There is the following relation: R(A1,A2,...,An). X is a submultitude of the multitude of attributes X $\subseteq$ {A1,A2,...,An}, Y is the complement {A1,A2,...,An}/ X. The relation R(A1,A2,...,An) can be written as follows: R(X,Y). The operation of **projection** of the relation R according to attributes X is marked as follows: **R[X]** and defined:

$$R[X] = \{x \mid \exists y : (x,y) \in R(X,Y) \}$$

2.2.Ø LOGICAL DEPENDENCES

2.2.1 FUNCTIONAL DEPENDENCE

There is relation R(A1,A2,...,An) and submultitudes of the multitude of attributes X $\subseteq$ {A1,A2,...,An} and Y $\subseteq$ {A1,A2,...,An}. R[XY] denotes the projection of the relation R according to attributes from X and Y. The

functional dependence X --> Y exists if and only if it is valid for R[XY] in every moment that there is a functional dependence R[X] --> R[Y].
The functional dependence X --> Y is said to be **complete** if for every real submultitude X' (X' ⊆ X) it is valid that X' -/-> Y. If it is valid that X' --> Y then the functional dependence X --> Y becames a **partial functional dependence.**

2.2.2 KEY

R(A1,A2,...,An) forms a relation. X which is a submultitude of the multitude of attributes is said to be the **key** of the relation R if and only if the following two conditions are fulfilled:

(i) X determines functionally all attributes of the relation R, X --> Ai for i = 1,...n.

(ii) no real submultitude of the multitude X possesses this characteristic, for X' ⊆ X X' -/-> Aj is valid 1<j<n.

If such X ⊆ {A1,A2,...,An} does not exist, that is valid X --> Ai, for i = 1,...,n, then the **key** of the given relation is a complete multitude of attributes A1,A2,...,An.

2.3.Ø NORMAL FORMS

Normal forms are rules on the joins of attributes into relations in which logical dependences are taken into account /DATE86/. Taking these rules into consideration the iregularities (anomalisms) of data input, deleting and updating are avoid.

2.3.1 THE FIRST NORMAL FORM

The relation R is in the **first normal form** if and only if the values in the domains are atomic for every attribute A in the relation R.

2.3.2 THE SECOND NORMAL FORM

Let X be the multitude of all attributes R(A1,A2,...,An), which are not a part of the key of the relation R. It is said that the relation R is in the **second normal form** if and only if each attribute from X is **completely functionally dependend** on each key of the relation R(A1,A2,...,An).

If the relation R(A1,A2,...,An) is not in the second normal form, then their exists a **decomposition** of the relation R(A1,A2,...,An) into a multitude of relations which are in the second normal form. The relations obtained in this way can be united again into a previous relation by means of the operation of natural join.
There is the relation R(A1,A2,...,An) which does not exist in the second normal form. The relation R can be recorded equivalently in the form R(X,Y,Z). In this case X means a multitude of key attributes and Y means a multitude of non-key attributes. X --> Y forms a partial functional dependence. The multitude Z covers all the remaining attributes of the relation R(A1,A2,...,An) which exist neither in the multitude X nor in the multitude Y.

The multitude X can be shown by X = X'X''. Then it is valid: X' --> Y. It is a complete functional dependence.

If the relation **R(X,Y,Z)** is supplanted by the projection **R[XZ]** and **R[X'Y]**, then the projection R[X'Y] is in the second normal form because X' --> Y forms **full functional dependence**. It should be found in which normal form the projection **R[XZ]** exists and, if necessary, this projection should be decomposed in the same way as **R(X,Y,Z)**. The procedure is final because after each decomposition of relation the relations with less number of attributes are obtained.

The previous statements can be confirmed by the following demonstration:

$$R[X'Y] *x' R[XZ] = R[X'Y] *x' R[X'X''Z] =$$

$$= R[X'YX''Z] = R[X'X''YZ] = R[XYZ] = R(X,Y,Z)$$

3.Ø APPLICATION OF PROLOG IN THE NORMALIZATION PROCES

3.1 NOTATION OF RELATION IN PROLOG

The description of structured of individual relations is presented by a relational schema. To model our relational schema the structured analaysis tools of DeMarco /DEMA79/ and Gane and Sarson /GANE79/ (data flowcharts, data dictionary, data store) were used. The result of modeling is the relational schemas wich are used for data storage by the relational database management system. The relation (the relational schema filled up with data) exits, depending on the choice of method in the first normal form or unnormalized.

The tables obtained were inrespective of their normal form, stored in the form of PROLOG structure. PROLOG's built-in database ensures a basic mechanism for data storage and data access. The notation of relation, that is of the whole relational base is a simple one. The data obtained from a relation are record in the form of PROLOG's facts, consisting of predicates and attributes. The name of relation is written in predicate; the values of attributes obtained from relational scheme are written in attributes. After the input of all data into the relational schema we can see that there is a record on the screen of the terminal which is equivalent to the record on the paper. This means that copying the relation from the paper to the screen is 1:1 (one-to-one).

3.2 IMPLEMENTATION OF PROJECTION BY MEANS OF PROLOG

The table into which the required columms were translated and in which the redundant lines were deleted is the result of operation **projection**.

The presented relations are described by means of PROLOG and in this way the program implementing a projection of suitable relation is designed.

```
* Program Projection *

projection(List_of_solutions,Solutions):-
find_equal(List_of_solutions,Solutions).

find_equal([],[]).

find_equal([H|T],Solutions):-
member(H,T),
find_equal(H,Solutions).

find_equal([H|T],Solutions):-
not_memeber(H,T),
write(T),
nl,
find_equal(T,[H|Solutions]).
member(X,[]):-!.
memeber(X,[_|Y]):- memeber(X,Y).
```

```
not_memeber(X,[]):-!.

not_memeber(X,[Y|Z]):- X \== Y,
                       not_memeber(X,Z).

findall(X,H,_):- asserta(found(mark)),
                 call(H),
                 asserta(found(X)),
                 fail.

findall(_,_,L):- collect_found([],M),
                 !,
                 L = M.

collect_found(S,L):- getnext(X),
                     !,
                     collect_found([X|S],L).
collect_found(L,L).

getnext(X):- retract(found(X)),
             !,
             X \== mark.
```

In the working version of the program Projection the implementation of the operation is released by the predicate **findall** and **projection**.

```
?-findall([chosen_atributes*],
  name_of_relation*(all_atributes*),Solutions)),
  projection(Solutions,M).
```

In the relation chosen all possible solutions are looked for, first. Then in the list of solutions (List_of_solutions) all redundant lines are excluded so that equal records (find_equal) are looked for which are not placed (not_member) on the list of final solutions (Solutions).

3.3.Ø CHECKING THE RELATION IN THE LIGHT OF ITS NORMAL FORM

3.3.1 UNNORMALIZED RELATIONS

According to the definition from Chapter 2.3.1, the relation exists in the first normal form if and only if the values in domains are atomic. That is the values in the domain are not lists or sets of values or composite values. In practice such records are found very often. If there is an unnormalized relation over wich we want to perform certain operations, then the relation should be translated into the first normal form.

For the normalization of such an unnormalized table the following is required:

- to check the existence of redundant lines and to exclude them,
- to find out whether the values of individual attributes in the domain are recorded as multitudes or lists. If such records exist, they should be translated into a corresponding form.

The program of record in PROLOG is based on checking the elements in the structure of list into which the previously written relation has been translated. When the number of elements in individual lists is found out, the paralel lying elements are joined into lines which are then transmited to the screen of the terminal in their normalized form.

---

* In the operation chosen_atributes, name_of_relation and all_atributes are substituted with the real names of atributes and relation.

```
* Program Normalization *

normalization(A):- A == tab,!.
normalization(A):- nonvar(A),
                   A \==  tab,
                   A =.. X,
                   write(A),
                   write('.'),
                   nl,
                   change(X),
                   read(A1),
                   normalization(.A1).

normalization(A):- read(A1),
                   normalization(A1).


change([H|Tail]):- P = H,
                   split(P,Tail,List_of_goals).

split(P,Tail,List_of_goals):-
count_arg(Tail,N),
count_el_arg(Tail,K),
[H|T]=Tail,
join_rest([H|T],N,1,S,K,1,List_of_goals),
form_goals(P,G,List_of_goals).
count_arg([],-1).
count_arg([G|T],N) :- count_arg(T,N1),
                      N is N1 + 1.

count_el_arg([H,[H1|[]]|T],1).
count_el_arg([H,[H1|X]|T],K):-
count_el_arg([H,X|T],K1),
K is K1 + 1.

join_rest(_,_,_,S,K,M,S1) :- M2 is K + 1,
                             M2 = M,
                             reverse_list(S,O),
                             S1=O,!.

join_rest(Tail,N,N1,List_of_goals,K,M,S1) :-
join_el(Tail,[],N,N1,M,S),
M1 is M + 1,
join_rest(Tail,N,1,[S|List_of_goals],K,M1,S1).

form_goals(_,_,[]) :- !.
form_goals(P,H,[H1|T1]) :-
reverse_list(H1,H2),
X =..[P,H,H2],
tab(3Ø),
write(P),
write('('),write(H),copy(G2),write(').'),
nl,
form_goals(P,G,R1).

copy([]) :- !
copy([G|R]) :- write(','),
               write(H),
               copy(T).

join_el(L1,L2,N,N1,M,List) :- N2 is N + 1,
                              N2 = N1,
                              List = L2,!.

join_el([H|T],T1,N,N1,M,S)  :-
append_link(T,N1,M,T2),
N2 is N1 + 1,
join_el([H|T],[T2|T1],N,N2,M,S).

append_link(T,N1,M,X) :- n_link(N1,T,Arg),
                         n_link(M,Arg,X).

n_link(1,[H|T],C) :- C = H, !.
n_link(N,[H|T],C) :- N1 is N - 1,
                     n_link(N1,T,C).

reverse_list([],[]).
reverse_list([H|T],L) :- reverse_list(T,L1),
                         append(L1,[G],L).

append([],L,L).
append([H|RT,L,[H|T1]) :- append(T,L,T1).
```

3.3.2.Ø STATING THE SECOND NORMAL FORM

In Chapter 2.3.Ø we pointed out that in inputting, deleting and updating irregularities can occur if normal forms are not taken into account. An irregular input can prevent to add new lines if all the values for attributes are not known. In deleting the lines the information on certain attributes can be lost while the updating of one attribute value requires a change in all relations where this attribute appears.

3.3.2.1 STATING THE LOCAL AND PARCIAL FUNCTIONAL DEPENDENCE

According to the definition in chapter 2.2.1, the existence of functional dependence is stated by checking the values of attributes that form the multitudes X and Y, between which we would like to shate the existence of functional dependence (X --> Y). It is valid that for each value x, x ¢ X there is exactly one y, y ¢ Y.

To state the functional existence by means of PROLOG the program Projection is used first to design the relation **R[XY]**. Then in this relation the condition **R[X] --> R[Y]**, according to chapter 2.2.1 is fulfilled. By means of a modified predicate find_equal from the program Projection equal x's are searched. If they exists then the value of belonging y's we can conclude to state the existence of functional dependence.

According to Chapter 2.2.1 the functional dependence can be a full or a partial one. To state this characterictic the multitude atribute X should be decomposed into real submultitudes (X', X'', X''',....). Then the existence of partial dependence between the real submultitudes (X', X'', X''',...) and the multitude of attributes Y should be checked according to the procedure described.

A rule to decompose the list (multitude X) into sub-list or elements (real submultitudes of the multitude X) should be added to the program in PROLOG.

3.3.3 NORMALIZATION OF RELATION INTO THE SECOND NORMAL FORM

The relation which was found out that it was not in the secon normal form should be decomposed into several tables that would meet the condition on second normal form.

According to the definition in Chapter 2.3.2, the relation R(A1,A2,...,An) is decomposed into two tables **R[XZ]** and **R[X'Y]** which form the result of the projection. The relation **R[X'Y]** exists in the second normal form because the following condition should be met: X' --> Y forms full functional dependence.
To normalize the relation into the second normal form given programs for projection and full functional dependence are used.

To check the relation according to its third normal form similar stepts (required for the third normal form) should be carried out.

4.Ø CONCLUSION

The described process of normalization in PROLOG is performed by IJS PROLOG on the main frame computer VAX 88ØØ. The local information system of research group was chosen as a model. The data on research group members, their activities, working results and the tools used by the member are recorded.

The fifth generation language **PROLOG** was chosen because of its simple copying of relation into the structure of PROLOG's facts and definition of rules (records of program) which implemented certain actions within the normalization process.

To bring the application nearer to the circle of potential users (also to those possessing no knowledge how to program) the application will be transfered to a Personal Computer by means of operation system DOS (TURBO PROLOG Borland, Inc. will be used). In this development phase, the application offers above all an aid for good design of relational database which provides the existence of successfull information system.

5.Ø LITKRATURK

COOD7Ø E.F.Codd: A Relational Model of Data for Large Shared Data Banks, Communication of the ACM, Volume 13, No.:6, June 197Ø

CLOC84 W.F.Clocshin, C.S.Mellish: Programming in Prolog, Springer-Verlag, Berlin 1984

DEYI84 Deyi Li: A Prolog Database System, Research Studies Press LTD,1984

DATE83 C.J.Date: Database: A Primer, Addison-Wesley Publishing Company, 1983

DATE86 C.J.Date: An Introduction to Database Systems, Volume 1, Addison - Wesley Publishing Company, 1986

KELL87 R.Keller: Expert System Technology Development and Appplication, Yourdon Press, Prentice-Hall Company Englewood Cliffs, NJ 1987

MARC86 C.Marcus: Prolog Programming, Addison-Wesley Publishing Company, 1986

WAH86 B.W.Wah, Guo-Jie Li: Tutorial: Computers for Artifical Intelligence Applications, IEEE 1986

A SIMULATION APPROACH BEFORE USING THE INDUSTRIAL MICROCOMPUTER CONTROLLER

INFORMATICA 3/1988

Anton Zorman
Maksimiljan Gerkeš
Viljem Žumer
Krista Rizman
Tehniška fakulteta Maribor

UDK 681.326.06:519.876.5

In this paper we describe a simulation packet that enables the verification of the software portion of the controller before implementating the controller on a real object. This program is unique of its kind, as far as we know, because it enables the examination of the controller's behavior in planner's workplace and so considerably reduces start expenses. The simulation packet enables controlled execution of the user's program, clearly arranged writing out of data on the screen or on the printer and gives a chance for data modification.

SIMULACIJSKI PRISTOP PRED UPORABO INDUSTRIJSKEGA MIKRORAČUNALNIŠKEGA KRMILNIKA.
V članku opisani **simulacijski paket** omogoča verifikacijo programskega dela krmilja pred vgraditvijo krmilnika na objekt. Po nam znanih podatkih je to edini tovrstni program, ki omogoča preizkus obnašanja krmilja na projektantovem delovnem mestu in s tem znatno znižanje zagonskih stroškov objekta. **Simulacijski paket omogoča** prikaz izvajanja uporabniškega programa, urejen in pregleden izpis vrednosti na zaslon ali na tiskalnik in možnost spreminjanja vrednosti podatkov.

1 Initial considerations about the industrial microcomputer controller and it's simulation

Requests for a system which upgrades standard programmable controller functions came from industry. Initial efforts were made by Metalna, Maribor. After it's definition phase the project was supported by the Research Society of Slovenia (Raziskovalna Skupnost Slovenije).

Intention and practical use of the industrial microcomputer controller

The controller is intended for control in capital equipment facilities plants, where heavy environment conditions and the immense equipment costs do not allow any compromise. A number of unique functions were built in the controller to obtain the required functionality. Special attention was dedicated to the user's program - control application development and verification tools.

The paper decribes a unique part of the user development and verification software which allows controlled application verification at the planner's workplace. This function reduces starting expenses when a capital object is put into work.

With convenient tools, software design methods and simulation, most of the mistakes, bugs and imperfections of the controller's software are discovered and removed before implementation on an object.

Thus, the simulation of the controller's software can be carried out in different ways. Module simulation is specially efficient. It allows that only verified software blocks are put together into larger structures. Typical software modules are functions and sobroutines like structures in high level language, and assembler like macros.

Simple use of the simulation packet is assured with it's **hierarchical tree structured menu**. The user selects from the menu on the screen the actions to be executed. Each action is determined with a function key on the keyboard. The user selects the desired action by pressing the adequate function key on the keyboard. Figure 1 shows an example of this principle.

METALNA Maribor
INDUSTRIAL CONTROLLER
SIMULATION

F1	F8
SIMULATION	END OF SIMULATION

Fig. 1: The initial simulation menu.

After selecting the adequate menu's window of a selected action, it lights up in yellow colour. This light is an advertisement for the user to notice which action he had selected. In the same way, the simulator advertises the user

when he returns back over the menues. If there is no special, objective reasons, the selected action is executed immediately.

The user can present his program for the controller eitheir in a graphic form, the so-called **contact networks**, or in textual, **mnemonic form**.

The user must call the analyser before applying the simulation of his program. The analyser is a kind of compiler. It was designed for this special purpose: it translates the user's program together with the declaration module and possibly with some other files for functions or subroutines into the 'object form' which is 'understood' and executed by the simulation, when requested.

The simulation can be executed only when the analyser does not find and report any error(s) in the user's program file, nor in the declaration module's file. A example of both files is on Figure 2a and Figure 2b.

```
MAIN
   SET   M1
   RESET M2
   BP    5
   ADD   CB12,     CB328,    B1
   SUB   CW7,      CW8,      W16
   MUL   CL323,    CL818,    L234
   W17 = CW17
   DIV   CW111,    CW71,     W12(W17), W20
   BP    9
   AND   W16,      W17,      W19
   CPL   CB12,     B12
   NEG   CB12,     B13
   BP    10
   RLC   CL328,    65,       L235
   SLC   CB115,    5,        B235
   TRANS CB115.3,  3,        L236.7
   TRANS M300,     100,      M250
   Q   =  M1
   JPQ   33
   NOP
33 L237  = L236
   M21   = N ( (M1 A M2) O (W16 LT CW87) )
   BP    20
   NOP
   BP    25
END.
```

Fig. 2a: The user's program (file TEST.MPR)

```
DECLAR
CONST
   CB10     <- 10
   CB12     <- 12
   CB115    <- 115
   CB119    <- 119
   CB123    <- 123
   CB321    <- 30
   CB328    <- 32
   CB378    <- 37
   CW17     <- 17
   CW71     <- 71
   CW73     <- 7300
   CW78     <- 78
   CW87     <- 87
   CW111    <- 111
   CW200    <- 2000
   CW312    <- 312
   CW419    <- 8841
   CW788    <- 788
   CL0      <- 0
   CL5      <- 5
   CL46     <- 46
   CL49     <- 49
   CL64     <- 64
   CL175    <- 175
   CL200    <- 2000000
   CL215    <- 215
   CL287    <- 287
   CL300    <- 3000000
   CL328    <- 328000
   CL818    <- 818000000
ENDCONST
 IOSPEC
    MOD1 is TY31
    MOD2 is TY31
    MOD3 is TY31
    MOD4 is TY31
BLOCK
   CS    11, YES,    70
   C     11, M6,     M7,  M8, M61,M62, M63, M64
   CS    12, NO,     80
   C     12, M6,     M7,  M8, M9, M10, M11, M12
   CS    13, YES,    90
   C     13, M10,    M11, M5, M6, M16, M17, M88
   TS    12, 10ms,   YES, 999
   T     12, M2,     M29, M3,     -
   TS    11, 100ms,  NO,  50
   T     11, M1,     M2,  M3,     M4
   TS    13, 1ms,    NO,  100
   T     13, M3,     M4,  M2,     M1
   TXS   23, READ,   2, "START"
   TX    23, M22,    M3
   TXS   24, WRITE,  2, "O.K."
   TX    24, M21,    M3
   TXS   25, NUMWRITE, 2, B333
   TX    25, M20,    M3
   TXS   26, NUMREAD,  2, L338
   TX    26, M26,    M3
   MS    13, 100ms,  NO,  333
   M     13, M2,     M20
   MS    14, 10ms,   YES, 238
   M     14, M3,     M27
   MS    15, 100ms,  NO,  669
   M     15, M27,    M84
   DS     8, 1s,     11,  8,   B
   D      8, M2,     M32, M33
   DS    16, 1s,     18,  16,  W
   D     16, M25,    M26, M37
   DS    32, 1s,     13,  32,  L
   D     32, M2,     M21, M30
   RS    16, FIFO,   22,  L
   R     16, M2,     M24, M32, M41, M52
   RS    17, LIFO,   30,  B
   R     17, M2,     M25, M35, M43, M53
   RS    18, FIFO,   20,  W
   R     18, M3,     M55, M55, M53, M57
ENDBLOCK
ENDIO
END.
```

Fig. 2b: The declaration module (file D2.DCM)

NOTE

The user does not need to write the file type (MPR for user's (or main) program or DCM for declaration module), because special purpose editors do this. File types are always hidden from the users!

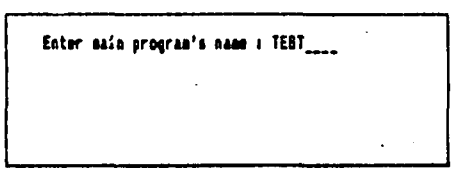


Fig. 3a: Reading a name of the main program.

After all neccessary files were translated, the simulation establishes the presence of error(s) very simply. If the analyser finds any error(s), then it does not produce the object file, which is a direct input to the simulation. In case of an error the simulator writes a message on the screen. In this message it tells the user that he can not execute the simulation because the error(s)

is(are) found and that the user can correct the error(s) in a corresponding editor.

The program **SIMULATION** first of all reads the name of user program's file (or main program's file) and the name of the declaration module. Figures 3a and 3b show the user's answers to both questions.

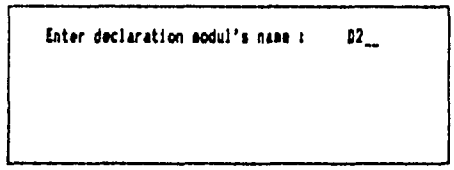


Fig. 3b: Reading the name of the declaration module.

After successful return from the analyser and before the simulation execution, the simulator writes a detailed report about constants and function blocks used in the declaration module (Figures 4a and 4b).

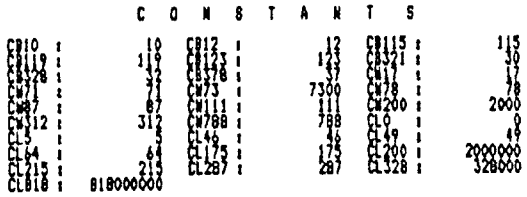


Fig. 4a: The report about CONSTANTS.

If the value of a constant is too large for it's data type, then the simulator assigns the largest positive or negative value to that constant, depended on its sign!
The data type of a constant is appointed by the second letter of the constant's name: **B**, **W** or **L** for byte (8 bits), word (16 bits) or a longword (32 bits).

The data type of an instruction operand identifies how many bits of storage should be considered as a unit and what is the interpretation of that unit to be. The simulator only recognises **the integer, BCD and ASCII data**. An integer can be stored in a bit, byte, word or in longword. Some instructions interpret the integer data as a signed value, while others as a bit strings.

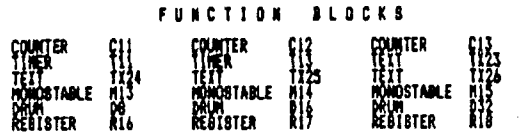


Fig. 4b: The report about FUNCTION BLOCKS.

1.1 Representation of a types of data and function blocks

The user can use the following types of data:

(a) CONSTANTS
The user assigns initial values to constants in the declaration module. During the execution of the simulation the simulator has read only access to it's value. The user can assign a new value to a constant in the later described menues **SINGLE CHANGE** and **ADJUST EDITOR**.

(b) SYSTEM DATA
The controller's system data are set up by the operating system. The simulator simulates this function by assigning them values. It can read only system data. The same principle to modify the system data is in valid as for constants.

(c) INTERNAL DATA
Internal data are general purpose data. The simulator can use them similarly as variables in a high level language: their values can be modified by the simulator.

(d) INPUT DATA
Input data are external physical inputs into the controller.

(e) OUTPUT DATA
Output data are physical outputs out from the controller.

We use **'the single assignment rule'** for all types of data, because of the simulation of parallel execution mutualy exclusive events.

Overview of the controller's data types

The user can use almost all combinations of types of data with data types in simulation. This survey is shown on Figure 5.

TYPE OF DATA	TYPE'S DATA NAME	TYPE'S DATA MARK	DATA TYPE'S NAME	DATA TYPE'S MARK	A SAMPLE
CONSTANT DATA:					
BYTE	Constant	C	Byte	B	CB18
WORD	Constant	C	Word	W	CW555
LONGWORD	Constant	C	Longword	L	CL234
SYSTEM DATA:					
BIT	System	S	Bit	*	S12
BYTE	System	S	Byte	B	SB373
WORD	System	S	Word	W	SW383
LONGWORD	System	S	Longword	L	SL947
INTERNAL DATA:					
BIT	Internal	*	Bit	M	M991
BYTE	Internal	*	Byte	B	B3
WORD	Internal	*	Word	W	W495
LONGWORD	Internal	*	Longword	L	L952
OUTPUT DATA:					
BIT	Output	O	Bit	*	O12.13
BYTE	Output	O	Byte	B	OB10.4
WORD	Output	O	Word	W	OW4.10
LONGWORD	Output	O	Longword	L	OL2.2
INPUT DATA:					
BIT	Input	I	Bit	*	I5.13
BYTE	Input	I	Byte	B	IB8.10
WORD	Input	I	Word	W	IW10.0
LONGWORD	Input	I	Longword	L	IL1.3

NOTE

Asterisk "*" means that at this place there is no mark!

Fig. 5: Survey about types of data.

The user has six **types of function blocks** besides the types of data mentioned above. The function blocks are:

(a) TIMER
The timer enables temporal control over events in an object. After a certain time delay something can happen, the value of which is programmable.

(b) MONOSTABLE
The monostable enables temporal control, too. It generates a pulse of specific duration, the value of which is programmable.
The main difference between the monostable and the timer is the following: the user can programmably control the timer through its inputs. After the user had enabled the monostable to start running, he can no longer programmably influence the monostable. Only one exception is allowed: the user can repeatedly start the monostable from the beginning!

(c) COUNTER
The counter permits the upcounting and downcounting of events. These two operations can be performed simultaneousy or not, as required.

(d) DRUM CONTROLLER
The drum controller enables temporal or event-driven (through its inputs) control: values of output bits of current drum step are assigned to actual bits. The two mentioned modes of operation are mutually exclusive.

(e) REGISTER
The register enables storage of data in two different ways:
* FIFO stack or
* LIFO queue.

(f) TEXT
The text enables simple input/output operations (communication between the user and the controller).

1.2 Simulation of a user's program execution

The simulation receives the user's program merged together with other files in object code on a file. The file has sequential organization. It consists of records arranged in the sequence in which they are written in the file (the first record written is the first record in the file, ... and so on).

Particular instruction needs more records. Records of the same instruction are always arranged in this way: a first record contains an operand which will have a result (one or two for division), a following record is a second operand, if it indeed exists in syntax of an instruction. After operands, if the instruction has any, comes the operator. This is an instruction which will be executed.

The simulator reads the records in the described regular sequence, too. Simulation of execution is based on the principle of **stack computer**. The simulator reads a record from the object code's file. Records are already in correct sequence, in so-called reverse Polish notation. The content of a record is either an operator or an operand.

If it is **an operand** then the simulator pushes it on the stack.
If it is **an operator** then the simulator pulls the corresponding number of operands from the stack, executes the operator (instruction) and assigns a value to a result.

The **IMCL** (Industrial Microcomputer Controller Language) is a mnemonic programmable language for our controller. The user can simulate all of the instructions of the **IMCL**:

ARITHMETIC OPERATIONS:
(1) ADD - arithmetic addition
(2) SUB - arithmetic substraction
(3) DIV - arithmetic division
Divide by zero:
The simulator assigns the largest positive or negative value to the result, dependebly on the numerator sign, a zero to the remainder and reports the overflow of the result.
The simulator writes the values of condition flags (**N**egative, **Z**ero, o**V**erflow and **C**arry) on the screen, then follow the messages about the mode of execution and about the current number of cycles - reiterations of execution of the user's program.
In case of dividing by zero the simulator always breaks execution and writes a message. After the message the user can continue with the execution of his program. He must press the key **RUN** - Figure 6!
(4) MUL - arithmetic multiplication

BIT OPERATIONS:
(5) A - logical AND operation over a bit's expressions
(6) O - logical OR operation over a bit's expressions
(7) N - negation of a bit's expression
(8) SET - bit set
(9) RESET - reset bit
(10) P - protection and assignement

BIT OPERATIONS BETWEEN TERMS (8, 16 or 32 bit string's length):
(11) OR - logical OR operation
(12) XOR - logical XOR operation
(13) AND - logical AND operation
(14) NAND - logical NAND operation
(15) NOR - logical NOR operation

COMPLEMENTS:
(16) NEG - one's complement
(17) CPL - two's complement

TRANSFER OF BIT STRING:
(18) SLC - shift left
(19) SRC - shift right
(20) RLC - rotate left
(21) RRC - rotate right
(22) TRANS - general purpose transfer of bits between bit strings

RELATIONAL OPERATORS:
(23) NE - operands are not equal ?
(24) EQ - are both operands equal ?
(25) LT - first operand is less than second one
(26) LTE - first operand is less than or equals to the second one
(27) GT - first operand is greater than second one
(28) GTE - first operand is greater than or equals to the second one

CONVERSIONS:
(29) CBIN - conversion from BCD to two's complement
(30) CBCD - conversion from two's complement to BCD

CONTROL OPERATIONS:
(31) JPQ - jump if Q bit is equal 1
(32) JPnotQ - jump if Q bit is equal 0
(33) JPX - jump if X bit is equal 1

(34) JPnotX - jump if X bit is equal 0
(35) JP - unconditional jump
The simulator writes a message to the user that the next step will be to execute a labelled program statement corresponding to the label of the JUMP statement. Jump skips the statements between JUMP instruction and this statement!
The simulator writes this message only in step-wise mode of execution!

CALL OPERATIONS:
(36) CALLM - calling a module
(37) CALLQ - conditional calling a module

MISCELLANEOUS OPERATIONS:
(38) BP - break point
When the simulator reaches a break point that it is not cancelled out in the user's program, it breaks te execution of simulation. The simulator writes a message about the break point irrespective of the mode of execution: step-wise mode or continous mode.
(39) EQUAL - assignement an operand's value to a result
(40) NOP - no operation
In the step-wise mode of execution the simulator writes only a message that it reached the NOP instruction and reassigns all condition flags to zero.

Instructions are **orthogonal** which means that the user can use the same instruction with different data types. For example: once with a byte, some other time with a longword.

Internal types of data, which are longer than one bit, we can **address** also in **index mode** and **indirect (deferred mode)**;
for example: **ADD (W3), W4(W6), W3.**
In such cases the value of **indirectly addressed internal variable** tells the simulator on which internal datum (variable) the instruction will be actually executed, or in **index mode of addressing**, (indexed variable is within round brackets) a sum of both internal variables' values gives index (**address**) of actual internal datum.

1.3 Representation of a user's program execution

You can repeatedly execute the simulation of yours program as many times as you like. Every time you can choose one mode from the following **modes of execution**:
(a) more cycles,
(b) single cycle,
(c) step-wise mode,
(d) continous mode,
(e) with break point(s) or
(f) without break point(s).

Some modes of execution are compatible with others. The user can, for example, execute the simulation in step-wise mode, with break points and has more cycles. One cycle is one iteration of the user's program execution.

The key **RUN** (Figure 6) enables a **commencement** or a **continuation** of user's program execution.

The **step-wise mode of execution** enables the user's program execution step by step, one instruction after another. For particular instructions a message is written on the screen. The message contains rudimentary informations: which instruction is executed, operand names and values and values of conditional flags (Negative, Zero, oVerflow and Carry).

The simulation can start with the following default modes of execution: a single cycle, continous mode and with break points! If the user does not choose the step-wise mode then the entire user's program is executed at least once (depended on the number of cycles - iterations of execution!) without the simulator writing out any message, except if there is a run time error or a break point instruction or a jump instruction!

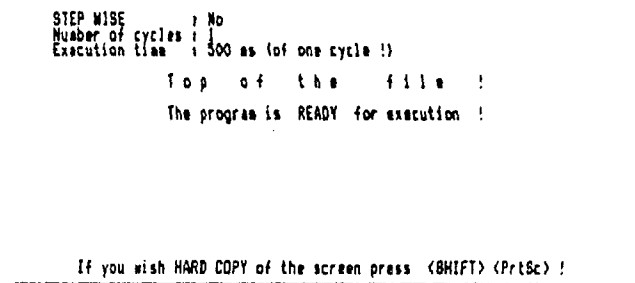


Fig. 6: A fundamental menu of simulation.

The user selects either the step-wise mode or the oposite continous mode, with the key **STEP WISE** (Figure 6). The step-wise mode and the continous mode are mutually exclusive. A new state is oposite to a previous state. At commencement of execution **the default state** is the **continous mode**, so if the user wishes the step-wise mode, he must press the key **STEP WISE** (Figure 6).

The key **MORE CYCLES** permits to inscribe the value of the number of cycles (Figure 7).

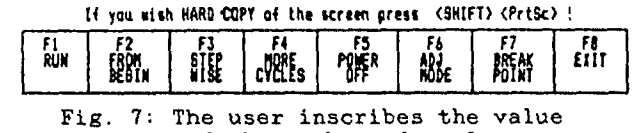


Fig. 7: The user inscribes the value of the number of cycles.

The user can cancel out all of the used **break points** or just some of them, or gives them active status, if he chooses a menu of break points (the key **BREAK POINT**), which is shown on Figure 8.

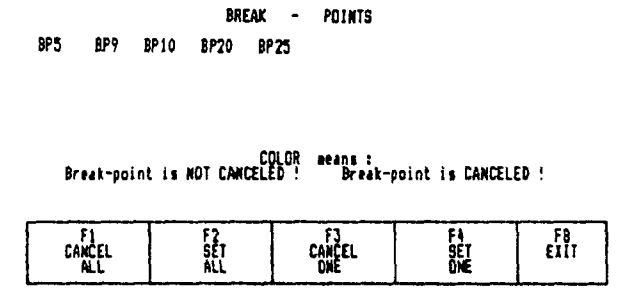


Fig. 8: A menu of break points.

If break point (BP) is cancelled out, then the execution of simulation is not broken!

Before executing an instruction, the simulator verifies if the neccessary operands have values. If they do not have, the simulator calls the user's attention to this fact. The user can choose either to inscribe an initial value to a datum or to confirm a simulation's proposal to assign a zero value to that datum.

The simulation enables all inscriptions of digits (numerical data) in:

(a) binary number system,
(b) octal number system,
(c) decimal number system and
(d) hexadecimal number system.

The user indicates the desired number system by inscribing the first character: **B**, **O** or **H** for binary, octal or hexadecimal number system. For decimal number system he does not inscribe any letter before digits!

During the inscription of an integer number, the simulator ignores the prohibited characters. For example: letters that do not have sense for the selected number system, or letters the values of which are larger than the basis of the number system decremented by one. Likewise, the simulator reports an error if the numerical value exceeds the allowed integer interval of datum's data type (a bit, a byte, a word or a longword).

During execution (depending on the selected modes of execution and menues) the user can make hard-copy of the screen, use **ADJUST MODE** that permits an overview and adjustment of used data and function blocks, select different modes of execution, cancel out or not cancel out break point(s) and, if he wishes, he can terminate the simulation. This function is enabled by the key **EXIT**, that always brings us into the previous menu.

The simulation permits the user to repeatedly execute different user's programs or combinations of the same user's program with different declaration modules.

2 An instance of step-wise execution of a simple user's program

Figure 9 shows an example of step-wise execution of a simple user's program, the source mnemonic form of which was presented on Figure 2a. We cut off the fundamental menu (presented on Figure 6) in order to spare space. Each message is written above this menu. Temporal order of messages is from top to the bottom of a paper.

```
                                          Flags :  N  Z  V  C
STEP WISE       : Yes                             0  0  0  0
Number of cycles : 5
Execution time   : 500 ns (of one cycle !)

                  Instruction : SET



Result(s)  :
              M1 :          1
```

```
                                          Flags :  N  Z  V  C
STEP WISE       : Yes                             0  1  0  0
Number of cycles : 5
Execution time   : 500 ns (of one cycle !)

                  Instruction : RESET



Result(s)  :
              M2 :          0
```

```
     The execution of the program is breaked at BP5 !
```

```
                                          Flags :  N  Z  V  C
STEP WISE       : Yes                             0  0  0  0
Number of cycles : 5
Execution time   : 500 ns (of one cycle !)

                  Instruction : ADD

Operand(s)  :
             CB12 :         12
            CB32 :         32

Result(s)  :
              B1 :         44
```

```
                                          Flags :  N  Z  V  C
STEP WISE       : Yes                             0  0  0  0
Number of cycles : 5
Execution time   : 500 ns (of one cycle !)

                  Instruction : SUB

Operand(s)  :
             CW7 :      19473
             CW8 :       8214

Result(s)  :
             W16 :      11259
```

```
                                          Flags :  N  Z  V  C
STEP WISE       : Yes                             0  0  1  0
Number of cycles : 5
Execution time   : 500 ns (of one cycle !)

                  Instruction : MUL

Operand(s)  :
            CL323 :    9591575
            CL818 :  818000000

Result(s)  :
             L234 : 2147483647
```

```
                                          Flags :  N  Z  V  C
STEP WISE       : Yes                             0  0  0  0
Number of cycles : 5
Execution time   : 500 ns (of one cycle !)

                  Instruction : =

Operand(s)  :
      CW17

Result(s)  :
             W17 :         17
```

```
                                          Flags :  N  Z  V  C
STEP WISE       : Yes                             0  0  0  0
Number of cycles : 5
Execution time   : 500 ns (of one cycle !)

                  Instruction : DIV

Operand(s)  :
            CW111 :        111
             CW71 :         71

Result(s)  :
          W12(W17) :          1   ( ---> W61 )
             W20 :         40   (remainder)
```

```
The execution of the program is breaked at BP9 !
```

```
                                                    Flags :  N  Z  V  C
STEP WISE       : Yes                                        0  0  0  0
Number of cycles : 5
Execution time   : 500 ns (of one cycle !)

                  Instruction : AND

Operand(s)  :
              M16 : 00101011 11111011
              M17 : 00000000 00010001

Result(s)   :
              M19 : 00000000 00010001
```

```
                                                    Flags :  N  Z  V  C
STEP WISE       : Yes                                        1  0  0  0
Number of cycles : 5
Execution time   : 500 ns (of one cycle !)

                  Instruction : CPL

Operand(s)  :
              CB12 : 00001100

Result(s)   :
              B12 : 11110100
```

```
                                                    Flags :  N  Z  V  C
STEP WISE       : Yes                                        1  0  0  0
Number of cycles : 5
Execution time   : 500 ns (of one cycle !)

                  Instruction : NEG

Operand(s)  :
              CB12 : 00001100

Result(s)   :
              B13 : 11110011
```

```
The execution of the program is breaked at BP10 !
```

```
                                                    Flags :  N  Z  V  C
STEP WISE       : Yes                                        0  0  1  0
Number of cycles : 5
Execution time   : 500 ns (of one cycle !)

                  Instruction : RLC

Operand(s)  :
              CL328 : 00000000 00000101 00000001 01000000
        Rotate no.  : 65 ( => 1 !!! )

Result(s)   :
              L235 : 00000000 00001010 00000010 10000000
```

```
                                                    Flags :  N  Z  V  C
STEP WISE       : Yes                                        0  0  0  0
Number of cycles : 5
Execution time   : 500 ns (of one cycle !)

                  Instruction : SLC

Operand(s)  :
              CB113 : 01110011
        Shift no.   : 5

Result(s)   :
              B235 : 01100000
```

```
                                                    Flags :  N  Z  V  C
STEP WISE       : Yes                                        0  0  0  0
Number of cycles : 5
Execution time   : 500 ns (of one cycle !)

                  Instruction : TRANS

Operand(s)  :
              CB115 : 01110011
        No. of bits : 3

Result(s)   :
              L236 : 00000010 00010101 01000111 00111011
```

```
                                                    Flags :  N  Z  V  C
STEP WISE       : Yes                                        0  0  0  0
Number of cycles : 5
Execution time   : 500 ns (of one cycle !)

                  Instruction : TRANS

Operand(s)  :
              M300 : 00000000 00000000 00000000 00000000  ...
        No. of bits : 100

Result(s)   :
              M250 : 00000000 00000000 00000000 00000000  ...
```

```
                                                    Flags :  N  Z  V  C
STEP WISE       : Yes                                        0  0  0  0
Number of cycles : 5
Execution time   : 500 ns (of one cycle !)

                  Instruction : =

Operand(s)  :
              M1

Result(s)   :
              Q :           1
```

```
The execution of the program is breaked at LABEL 33 !
```

```
                                                    Flags :  N  Z  V  C
STEP WISE       : Yes                                        0  0  0  0
Number of cycles : 5
Execution time   : 500 ns (of one cycle !)

                  Instruction : =

Operand(s)  :
              L236

Result(s)   :
              L237 :     34948923
```

```
                                                    Flags :  N  Z  V  C
STEP WISE       : Yes                                        0  1  0  0
Number of cycles : 5
Execution time   : 500 ns (of one cycle !)

                  Instruction : A

Operand(s)  :
              M2 :           0
              M1 :           1

Result(s)   :
        Result  is  :        0
```

```
                                                    Flags :  N  Z  V  C
STEP WISE       : Yes                                        0  1  0  0
Number of cycles : 5
Execution time   : 500 ns (of one cycle !)

                  Instruction : LT

Operand(s)  :
              M16 :       11259
              CM87 :         87

Result(s)   :
        Result  is  :        0
```

```
STEP WISE         : Yes                               Flags :  N  Z  V  C
Number of cycles : 5                                           0  1  0  0
Execution time    : 500 ms (of one cycle !)
                    Instruction : O
Operand(s)  :
       Stack - TOP :             0
       Stack - TOP :             0
Result(s)   :
       Result  is  :             0
```

```
STEP WISE         : Yes                               Flags :  N  Z  V  C
Number of cycles : 5                                           0  0  0  0
Execution time    : 500 ms (of one cycle !)
                    Instruction : N
Operand(s)  :
       Stack - TOP :             0
Result(s)   :
       Result  is  :             1
```

```
STEP WISE         : Yes                               Flags :  N  Z  V  C
Number of cycles : 1                                           0  0  0  0
Execution time    : 500 ms (of one cycle !)
                    Instruction : =
Operand(s)  :
       Stack - TOP
Result(s)   :
       M21 :                     1
```

```
The execution of the program is breaked at BP20 !
```

```
STEP WISE         : Yes                               Flags :  N  Z  V  C
Number of cycles : 5                                           0  0  0  0
Execution time    : 500 ms (of one cycle !)
                    Instruction : NOP
```

```
The execution of the program is breaked at BP25 !
```

```
STEP WISE         : Yes
Number of cycles : 5
Execution time    : 500 ms (of one cycle !)
          Start of execution of THE FUNCTION BLOCKS.
```

```
STEP WISE         : Yes
Number of cycles : 5
Execution time    : 500 ms (of one cycle !)
          E n d   o f   c y c l e  : 5

                    .
                    .
                    .
```

```
STEP WISE         : Yes
Number of cycles : 0
Execution time    : 500 ms (of one cycle !)

          E n d   o f   t h e   f i l e   !

          Execution of the program is finished !
```

Fig. 9: A step-wise mode of execution of a simple user's program.

3 Adjustment and writing out values

In the fundamental simulation menu (Figure 6), the user must select **ADJUST MODE** (Figure 10). The simulator permits two modes of adjusting and writing out data.

```
If you wish HARD COPY of the screen press  <SHIFT> <PrtSc> !
```

F1 CONT VIEW	F2 PROG VIEW	F3 DRUM EDITOR	F4 PRINT ALL	F5 ADJ EDITOR	F6 SINGLE CHANGE	F8 EXIT

Fig. 10: A menu of ADJUST MODE.

The first mode is directed to a particular datum or function block and so we call it **SINGLE CHANGE** (Figure 11). The second mode is directed to all values of the same type of data - **ADJUST EDITOR** (Figure 12).

```
If you wish HARD COPY of the screen press  <SHIFT> <PrtSc> !
```

F1 CONSTANT MONOSTABLE	F2 SYSTEM VARIABLE REGISTER	F3 INTERNAL VARIABLE COUNTER	F4 INPUT TIMER	F5 OUTPUT DRUM	F8 EXIT TEXT

Fig. 11: A menu of SINGLE CHANGE.

```
          CWO              Radix 8                  Constant Word
      0       1       2       3        4        5       6        7
 0
 1    -       -       -       -        -        -       -     000021
 2    -       -       -       -        -        -       -        -
 3    -       -       -       -        -        -       -        -
 4    -       -       -       -        -        -       -        -
 5    -       -       -       -        -        -       -        -
 6    -       -       -       -        -        -       -        -
 7    -    000107     -    016204      -        -       -        -
 8    -               -                -        -       -     000127
 9    -       -       -       -        -        -       -
10    -       -       -       -        -        -       -        -
11    -    000157     -       -        -        -       -        -
12    -               -       -        -        -       -        -
   You can use keys on the Numeric Keypad!    Help: press <Alt> H
```

F1 FILE TOP BOTTOM	F2 PAGE UP DOWN	F3 PAGE LEFT RIGHT	F4 PAGE TOP BOTTOM	F5 LINE UP DOWN	F6 LINE BEGIN END	F7 CHANGE VALUE PRINT	F8 EXIT LINE ?

Fig. 12: A menu of ADJUST EDITOR.

The way of selection is identical for both modes, because it runs through similar menues. The user selection starts on menu of types of data (Figure 13a). There are constants, internal data, system data, input and ouput data. The user chooses between a bit, a byte, a word and a longword in the next menu of data types (Figure 13b). The user can choose all data types for any selected type of data except a bit for constants, because the simulator does not have a constant bit!

If you wish HARD COPY of the screen press (SHIFT) (PrtSc) !

F1 CONSTANT	F2 SYSTEM VARIABLE	F3 INTERNAL VARIABLE	F4 INPUT	F5 OUTPUT	F8 EXIT

Fig. 13a: Menu of types of data.

If you wish HARD COPY of the screen press (SHIFT) (PrtSc) !

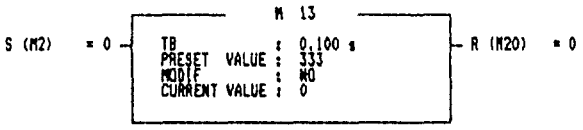
F1 BITS	F2 BYTES	F3 WORDS	F4 LONGWORDS	F8 EXIT

Fig. 13b: Menu of data types.

3.1 Adjustment and writing out values of function blocks' data

The menu **SINGLE CHANGE** (Figure 11) enables adjustment and writing out values of function blocks' data.

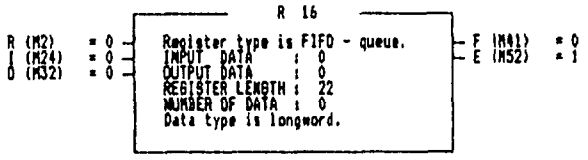
After selecting the type of a function block, the user inscribes its successive number and then, if it actually exists, the simulator draws its picture on the screen. The simulator then writes important information about it: type of function block, its successive number, names and values of its input and output bits and current values of its typical variables. In a menu, under each drawing, the user can read which variables he can adjust and what actions he can use (Figures 14a, 14b, 14c, 14d, 14e and 14f).



If you wish HARD COPY of the screen press (SHIFT) (PrtSc) !

F1 TIME BASE	F2 CURRENT VALUE	F3 PRESET VALUE	F8 EXIT

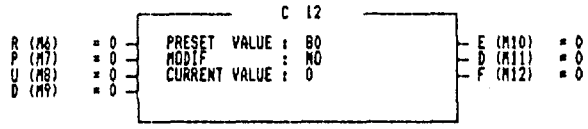
Fig. 14a: A menu of MONOSTABLE.



If you wish HARD COPY of the screen press (SHIFT) (PrtSc) !

F1 INPUT DATA OUTPUT DATA	F2 REGISTER LENGTH NUMBER OF DATA	F8 EXIT CONTENTS

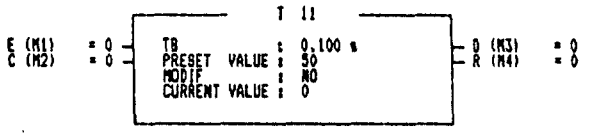
Fig. 14b: A menu of REGISTER.



If you wish HARD COPY of the screen press (SHIFT) (PrtSc) !

F1 CURRENT VALUE	F2 PRESET VALUE	F8 EXIT

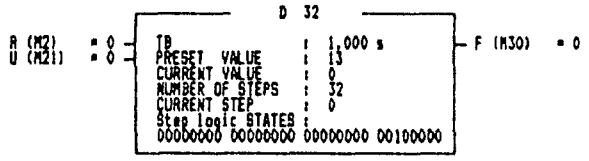
Fig. 14c: A menu of COUNTER.



If you wish HARD COPY of the screen press (SHIFT) (PrtSc) !

F1 TIME BASE	F2 CURRENT VALUE	F3 PRESET VALUE	F8 EXIT

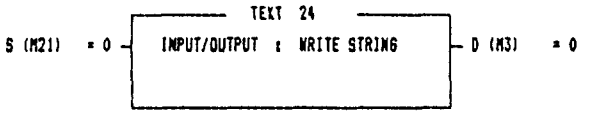
Fig. 14d: A menu of TIMER.



If you wish HARD COPY of the screen press (SHIFT) (PrtSc) !

F1 TIME BASE	F2 CURRENT VALUE	F3 PRESET VALUE	F4 NUMBER OF STEPS	F5 CURRENT STEP	F8 EXIT

Fig. 14e: A menu of DRUM.



Your message is O.K. .

If you wish HARD COPY of the screen press (SHIFT) (PrtSc) !

F1 INTERNAL VARIABLE UNIT 1	F2 MESSAGE UNIT 2	F8 EXIT

Fig. 14f: A menu of TEXT.

3.2 Adjustment and writing out values of data

As mentioned above, the user can adjust and write out values of data in two places in our simulation.

3.3 SINGLE CHANGE

After selecting type of data and its data type, the user inscribes its successive number in menu **SINGLE CHANGE**. If the entered name is correct in the context of Figure 5 and has a value, then the simulator writes out its value in binary, octal, decimal and hexadecimal number system. Now, the user can adjust the value or return to the menu of data types. An instance of choosing and adjusting of a datum **CW200** in menu **SINGLE CHANGE** is shown on fugures 15a and 15b.

Constant Word NUMBER : 200__

It must be greater than -1 and less than 1000 !

F1 BYTES	F2 WORDS	F3 LONGWORDS	F8 EXIT

Fig. 15a: Choosing a datum.

```
                    CW200 -- Constant Word

OLD   VALUE :                          NEW   VALUE :
    Binary  : 0000 0111  1101 0000         Binary  : 1111 1000  0011 0000
    Octal   : 003720                       Octal   : 774060
    Decimal : 2000                         Decimal : -2000
    Hex     : 07D0                         Hex     : F830

          Is new nuaber's value correct ?
          Answer with   Y or N  !
```

F1	F2	F3	F8
BYTES	WORDS	LONGWORDS	EXIT

Fig. 15b: Adjusting the same datum.

3.3.1 ADJUST EDITOR

ADJUST EDITOR is a screen editor that enables overlooking of all values of a selected type of data. We can not usually write all the values on a screen at the same time. We can see those values which are not momentary on the screen by moving the window through a table of values (we called it a file sometimes). Function keys of the menu of **ADJUST EDITOR** enable this moving (Figure 16).

```
<Home> . . . . . To  Left  on line
<End> . . . . . To  right on line
<PgUp> . . . . . Page up
<PgDn> . . . . . Page down
<Up Arrow> . . . . . Line up
<Down Arrow> . . . . . Line down
<Left Arrow> . . . . . Bit  left
<Right Arrow> . . . . . Bit  right
<CTRL> <Home> . . . . . To top    of page
<CTRL> <End> . . . . . To bottom of page
<CTRL> <PgUp> . . . . . To top    of file
<CTRL> <PgDn> . . . . . To bottom of file
<CTRL> <Left Arrow> . . . . . Word  left
<CTRL> <Right Arrow> or <Tab> . Word  right

You can use keys on the Numeric Keypad!   Help: press <Alt> H
```

F1	F2	F3	F4	F5	F6	F7	F8
FILE TOP BOTTOM	PAGE UP DOWN	PAGE LEFT RIGHT	PAGE TOP BOTTOM	LINE UP DOWN	LINE BEGIN END	CHANGE VALUE PRINT	EXIT LINE ?

Fig. 16: The possibilities of ADJUST EDITOR.

We can move
- * to top of a file and
- * to bottom of a file.

We can move window
- * up,
- * down,
- * left and
- * right.

Within a window we can move
- * to top of a page and
- * to bottom of a page.

We can move a line
- * up and
- * down

and inside of the line
- * to left on line (beginning of line),
- * to right on line (end of line)

and from one datum to another
- * a datum left and
- * a datum right.

We can move out of a current window only by moving a window, because by moving within a window we can not come out of it.

Such moving is often time-consuming and as we wished to increase the efficincy of the **ADJUST EDITOR**, we added the above described possibilities still another one: the movement to a datum required by the user (the key **LINE ?** on Figure 16).

Besides these, function keys, we can also use the keypad's keys for the same purpose.

The key **CHANGE VALUE** permits adjustment of values and the key **PRINT** enables writing out all values of the selected type of data, clearly arranged on the printer.

4 Conclusion

The presented simulation packet is written in the programming language Turbo Pascal. The user can execute it on all personal computers **IBM XT/AT** type, or compatible, equiped with an operating system **MSDOS**.

The simulation packet is a composition of twelve independent programs, which are connected with standard procedure Execute. The number of source program lines exceeds 50000, the object code is more than 360k bytes of lenght.

The **SIMULATOR** is an integral part of the user's development tools. Consequently, the entire project offers the special purpose environments to the users.

5 References:

- * The industrial microcomputer controller's architecture (in slovene), Metalna Maribor, 1987,
- * V. Zumer et al., Workstation for software development, Technical report - in slovene, 1986
- * V. Zumer et al., Workstation for software development, Technical report - in slovene, 1987
- * Programmable Controllers SIMATIC S5 (different variants), SIMENS, 1984
- * Programmable Controllers TSX (different variants), Telemecanique, 1986
- * R.E Fairley, Software Engineering Concepts, McGraw-Hill Book Company, 1985
- * VAX Architecture Handbook, Digital Equipment Corporation, 1981

UDK 519.72

Anton P. Železnikar
Iskra Delta

The first part of this article deals with an intuitive background and with a coarse symbolic introduction of the subject called informational logic (or logic of information). In this context it examines the notion and possibilities of formalization of informational inference (reasoning) as a foreground of the arising logical discourse. First, as a representative of the class of all possible informational operators, the notion of the operator of informational arising (or of Informing) is determined, covering several informational necessities and possibilities of an infix, prefix, and postfix operator, of parallelnes and serialness of informational processes, and of its arbitrary semantic particularization and universalization. In this respect, the informational operator of arising $\models$ can be abstracted (particularized and universalized) according to the needs and goals of its application. Two types of operators of general Informing ($\models$ and $\nmodels$) and general non-Informing ($\not\models$ and $\not\nmodels$) are introduced, to enable also a description of some (reduced) parallel informational cases in a general (linearized) form. Further, the notion of the so-called informational cycle as an autonomous informational unit is introduced, so that operators of cyclical Informing ($\vdash$ and $\dashv$) and non-cyclical Informing ($\not\vdash$ and $\not\dashv$) are meaningful. Some cyclical informational schemes or scenarios are discussed within informational systems S1 to S9. Within the philosophy of the informational cycle some modal properties of informational operators are pointed out concerning counter-Informing and embedding of counter-information (systems S8 and S9). The next, currently concealed and unsufficiently revealed informational phenomenon is parallelism. In a symmetrical way to general and cyclical operators of Informing, for future axiomatical expressing of parallel informational processes the introduction of adequate parallel informational operators becomes necessary. In this sense, two sets of parallel operators are proposed, namely the cyclical parallel ($\Vdash$, $\dashv\!\!\!\dashv$, $\not\Vdash$, $\not\dashv\!\!\!\dashv$) and the general parallel ones ($\models\!\!\!\models$, $\nmodels\!\!\!\nmodels$, $\not\models\!\!\!\not\models$, $\not\nmodels\!\!\!\not\nmodels$). It is evident that these parallel operators need additional ones which will enable the expressing of communication and interaction among parallel informational processes. The principle of operational particularization and universalization represents the last point of this part of the essay. In the process of formation and transformation of informational formulae, informational operators can be particularized and universalized (semantically determined) due to the needs and goals of an arbitrary informational application. Again, all of the 16 proposed informational operators proposed are listed and described. At the end, two examples of a scenario of questioning (a discourse concerning information, appearance of a question, processes of questioning and answering, and an answer) are presented in the notation using informational operators.

Informacijska logika I. Prvi del tega članka se ukvarja z intuitivnim ozadjem in z okvirnim simboličnim uvajanjem predmeta, ki se imenuje informacijska logika (ali logika informacije). V tej povezavi se raziskujejo pojem in možnosti formalizacije informacijskega sklepanja kot predpogoji za nastajajoči logični diskurz. Najprej se opredeljuje pojem operatorja informacijskega nastajanja (ali informiranja) kot predstavnika razreda vseh možnih informacijskih operatorjev, ki hkrati zadošča različnim informacijskim potrebam in možnosti infiksnega, postfiksnega in prefiksnega operatorja, paralelnosti in serailnosti informacijskih procesov in poljubne semantične partikularizacije in univerzalizacije. Tako je mogoče informacijski operator nastajanja $\models$ abstrahirati (partikularizirati in univerzalizirati) skladno s potrebami in cilji njegova uporabe. Uvajata se dve vrsti operatorjev za

splošno informiranje ($\models$ in $\dashv$) in za splošno neinformiranje ($\not\models$ in $\not\dashv$), tako da je omogočeno tudi opisovanje nekaterih (reduciranih) paralelnih informacijskih primerov v splošni (linearizirani) obliki. Nadalje se uvaja pojem t.i. informacijskega cikla kot avtonomne informacijske enote in operatorji cikličnega informiranja ($\vdash$ in $\dashv$) in necikličnega informiranja ($\nvdash$ in $\not\dashv$) postanejo tako smiselni. Obravnavajo se nekatere ciklične informacijske sheme ali scenariji, in sicer v okviru informacijskih sistemov S1 do S9. V okviru filozofije informacijskega cikla so prikazane nekatere modalne lastnosti informacijskih operatorjev, ki opisujejo protiinformiranje in vmeščanje protiinformacije (sistema S8 in S9). Naslednji, trenutno še prikriti in nezadostno pokazani informacijski pojav je paralelizem. Simetrično k splošnim in cikličnim operatorjem informiranja so uvedeni ustrezni (eksplicitni) paralelni operatorji, ki naj bi omogočali aksiomatsko izražavo paralelnih informacijskih procesov. V tem smislu sta predločeni dve množici paralelnih operatorjev, in sicer ciklično paralelni ($\Vdash$, $\dashv\!\!\dashv$, $\nVdash$, $\not\dashv\!\!\dashv$) in splošno paralelni ($\models\!\!\models$, $\dashv\!\!\dashv$, $\not\models\!\!\models$, $\not\dashv\!\!\dashv$). Očitno bo tem paralelnim operatorjem potrebno dodati še operatorje za izražavo komunikacije in interakcije med paralelnimi informacijskimi procesi. Princip operatorske partikularizacije in univerzalizacije je opisan v zadnjem poglavju tega dela eseja. V procesu formiranja in transformiranja informacijskih formul je mogoče informacijske operatorje partikularizirati in univerzalizirati skladno s potrebami in cilji neke poljubne informacijske aplikacije. Tako so ponovno našteti in pojasnjeni vsi predloženi informacijski operatorji. Na koncu eseja sta prikazana dva primera scenarijev vpraševanja (diskurza, ki obravnava informacijo, pojavitev vprašanja, proces vpraševanja in odgovarjanja in odgovor), in sicer v notaciji predloženih informacijskih operatorjev.

P A R T O N E :

A N I N T U I T I V E - S Y M B O L I C B A C K G R O U N D O F I N F O R M A T I O N A L L O G I C

0. Introduction

Some brain-damaged human beings, like computers, deal with familiar domains in that completely logical manner. Patients suffering from a neurological disorder called "agnosia" exhibit a total dependence upon analysis and rational explanation. For them everything must be decomposed into features and relationships before it can be understood. For example, a victim of agnosia presented with a triangular object will first report that it has three angular corners connected by straight sides and only then will conclude that it is a triangle. Restricted to understanding of only this kind, the patient is unable to function adequately in the everyday world.

Hubert L. Dreyfus and
Stuart E. Dreyfus [1] 64

0/0

This part of the essay represents an intuitive and roughly symbolic introduction into the possibilities of informational objects and operations which will constitute a kind of logic. There are appeared and will appear several objections stigmatizing the so-called informational principles [4] and particularly trials to develop a discipline called informational logic [6] as a special and extended logical system. Maybe, the strict mathematical doctrine and algorithmically based computer science will depreciate these trials, concluding that they belong to the domain of science-fiction. They will traditionally assume that it is not possible to change the principles which have been arising for thousands of years and have been approved as a doctrine and as a practice. Had there really been so, when the purposes and philosophies of new development, constructive possibilities, and human mind have been changed dramatically too?

Modal logic has already taken its triumphant course in the field of artificial intelligence. As a tool of research, it is able to capture the domain of various "non-scientific" objects to which beliefs, awarenesses, and reasonings of human mind may belong. Of course, modal logic as a system of necessity and possibility does not concern only the "must be" and the "may be", but also various other (imaginable, psychological) modal operations (viz. "believe", "know", "reason", "understand", etc.). As the research in this essay will show, informational operators will possess (merely a kind of) modal and a regular operational nature and, in this sense, will remain in the investigational realm of commonsense reasoning.

0/1

Briefly, informational logic could be understood as the logic of informational existence, potentiality, and phenomenology, all of which root in spontaneous informational arising. It is the logic concerning not only information processing in living organisms, but

also the coming of information into existence and using this information for processing of life and survival. Such a logic, as presented on lingual (extroverted) level, needs a metalinguistic power of different languages hidden in their composite verbal or even sentential expressiveness. In this way, this essay will be a study of formal informational logic, raising a substantial number of questions relating not merely to the philosophy of information, but also to other philosophical realms.

Informational logic is a new term and represents the marker for a particular logical system that arises from the concepts of information, as they were determined by the author [2, 3, 4, 5, 6]. Informational logic uses its own language with characteristic syntax and semantics. This language is trying to capture the most general, but also the most characteristic notions of the philosophical and mathematical logic, as well as some particular notions of the informational arising as it is understood from the philosophy of information.

When possessing its own language, informational logic constitutes a formal system with its own logical calculus (or axiomatic basis in the generative sense). For the purpose of the expressiveness of informational logic and informational calculus, a set of standard and non-standard signs will be introduced. However, some of the standard signs will obtain additional, generally more flexible meanings, although they will retain their particular ones.

1. The Notion and Formalization of Informational Inference

... we do not yet (and may never) know whether there is any proof that is elegant, concise, and completely verifiable by a (human) mathematical mind.

Kenneth Appel and
Wolfgang Haken [7] 162

1/0

At the construction of a logical system, symbols and rules for combining symbols into formulae and for manipulating these formulae have to be set up. Further, meaning has to be given to these symbols and formulae. An axiomatic basis of a logical system consists of primitive symbols, formation rules, axioms, and transformation rules. The well-formed formulae (wff) obtained by application of transformation rules are called theorems. A thesis of a given system is either an axiom or a theorem. Thus, an informationally interpreted logical system can be obtained. Within such a system, questions of adequate information concerning informational arising and other informational possibilities, for instance of truth and falsity, can be raised.

1/1

Truth and falsity belong to the main polar categories in philosophy and particularly in logic. In a two-value formal logic, propositions of the logical calculus are estimated for truth and falsity. In a multivalue formal logic there exist intermediate values of propositions, ranging from the absolute truth to the absolute falsity. However, it has to remain in the foreground of our discourse that truth and falsity are only very particular and also substantially and abstractly narrowed forms of information, even in those cases where they are semantically dispersed between their absolutes. In many theoretical and practical cases truth and falsity can simply be shifted outside of the field of relevance.

Informational truth can be a generative set of informational cases representing various informational forms and processes concerning truth. Similarly, informational falsity can be an adequate generative set of informational cases which are estimated to support falsity. These two generative sets can in no case be disjoint. As they appear within living information, truth and falsity can depend essentially on beliefs, awarenesses, reasonings, etc. of living informational agents. In general, truth and falsity are coming into existence as information, which arises from case to case and is informationally observed, estimated, investigated, and understood as truth and falsity according to various valid, counterfeit, and contrariwise forms and processes of information.

1/2

As pointed out, informational inference will deal with a sort of adequacy in cases of concluding, reasoning, and oriented thinking. Informational inference will embrace processes of Informing in the broadest sense. Although Informing by itself will be understood as informational arising, the informational arising by itself will be understood as the most basic informational regularity. And what is the observed regularity other than a kind of logic? The process of Informing will be formalized by steps of informational inference and, within these steps, information will regularly arise from one informational step to another.

In this respect, in a process of Informing the so-called derivative steps of truth and falsity will appear as special cases of inference. Informational logic will deal with much more general informational cases as truth and falsity could be by themselves. This logic will be derived from the principles of information [4], considering informational circularity (recurrence), spontaneity, arising, counter-Informing, embedding, etc. Since all these cases are merely particularities of informational arising, the arising itself will be denoted by the unique and specific symbol $\models$. Or more precisely, several families of specific symbols, denoting informational arising in a

general sense, in the framework of the so-called informational cycle, and in the domain of the informational parallelness will be used. It will be shown how informational operators with a general meaning can be specialized (or even additionally universalized), so that they adopt a particular function. In this manner, the known and common logical, functional, set-theoretical, and other operators can be replaced by the particularized informational operators. In the sections following, several different cases of the arising symbols (operators) or symbols of arising will be revealed.

2. The Operator of Informational Arising

2/0

The goal of the following sections is to introduce the most general notion of informational arising. As we have learned in [6], the informational arising can be understood as a parallel-serial sequence of informational cycles. Information is arising spontaneously in a cyclical manner. It means that it is arising parallel-serially, cyclically, and spontaneously within a cycle itself, so that each informational cycle has its characteristic informational subcycles or cyclical parallel-serial steps (subarisings).

Instead of informational arising often the term Informing will be used. Arising is nothing else but the Informing of information. The term Informing can be used also for denoting the so-called informational steps (derivations) occurring within a cycle (processes of sub-Informing). The production of counter-information can be denoted as the counter-Informing and the process of the embedding of counter-information into the original or source information as the informational embedding (or simply embedding). However, Informing can be used also for denoting the entire arising-embedding cycle or also a parallel-serial sequence of several informational cycles. In this manner, Informing is the most general term to denote the processes of informational acting (arising, embedding, Informing, counter-Informing, etc.).

It would be reasonable to introduce a universal operator of Informing (functor, quantifier, relator, deriver, deducer, circulator, generator, creator, etc.) or even several different families of operators of Informing. As mentioned in [3], Informing has its semantical origin in the verb 'inform'. This verb incorporates the meanings of all possible verbs, sentences, and their linguistic combinations and can be used for the substitution of any operating information. The verb 'inform' represents the generative set of all possible semantical values, also potential ones, which can appear within a natural language or even in a language of thought or of mind. Moreover, in the same way as information denotes any information, the most simple and also the most complex one, the verb inform can represent any operational (relational, functional) information, i.e., also any combination of notions and their understandings. It means that semantically Informing as informational process is actively and passively operative in any possible, yet impossible, or arbitrary (spontaneous) way. In this sense under operational information each instructive, imperative, transformative, arising-transformational, information-changeable form of operation will be understood.

As logic does not concern necessarily only the problems of truth and falsity, informational discourse does not investigate merely information belonging to the generative sets of informational truth and informational falsity. The most general informational truth is that information arises, that it informs and counter-informs, that in its Informing it is circular (cyclic) and spontaneous, that counter-information has to be informationally embedded otherwise it ceases as informational noise, that two informational cases are always different, etc. In logic we say that "something is true" while in informational logic we will say that "something informs". Similarly, in logic we say that "something is false", and in informational logic we will say that "something informs in another or different way".

In a similar way that ordinary logic concentrates its investigations around the notions of truth and falsity, the informational logic will concentrate them around the notion of Informing, i.e., around the notions concerning information, Informing (of information), counter-Information, counter-Informing, information of embedding, informational embedding, informational arising (spontaneity), informational circularity (recurrence), etc. The goal of informational logic will be to relate and to connect essential informational items, pieces, and cases and to introduce an effective symbolism (formalization) for the purposes of the already developing future informational calculus. The main problem of this task will be to keep the concept of information open, i.e., not to close the discourse of informational logic and its formalism, but to enable further development of informational logic and the increasing of its investigational power.

2/1

In [6] we have introduced the symbol of arising A of information i by the expression Ai (maybe, in this case, the notation $\exists i$ would be more appropriate). In this expression A was classified as a sort of quantifier (informational operator), similarly to the notion of quantifiers in mathematical logic (for instance $\forall$ and $\exists$). The quantifier A could be similar, for instance, to the logical quantifier of existence (for instance, the informational arising is the coming of information into existence). It can be comprehended that the difference between existential and arising quantifier is nothing if not philosophical. The existence of information is subjected to the informational arising, so that the arising is not only a more precise, but also a more complex notion of informational existence, including the

existing, changing, occurring, coming into existence, ceasing, vanishing, disappearing of information, etc. Arising denotes the coming of information into existence and not only the existence of unchanged, and spontaneously non-arising and non-arisen information.

2/2

According to our discussion concerning the operator A, we have to define a new, complex, and general operator (not only a quantifier) of arising. In the informational calculus this operator will be used to replace the most general, but also the most particular operation. In philosophical, two-value logic, for instance, the so-called relational symbols of truth ($\models_T$) and falsity ($\models_F$) will be introduced. By these symbols, it is possible to create the so-called 'deductive' chains (calculations) when proving or deriving (by means of transformation rules) theorems from given set of axioms. In this sense, relations $\models_T$ and $\models_F$ are used as a sort of operations stepping from one derivation to another (in a deduction chain), within a logical system.

DF1 The most general operator of informational logic will be the sign $\models$, representing the broadest (the most complex, parallel, circular, and spontaneous) meaning of Informing (or of arising) of information. In general, if α is information, the notations

$$\alpha \models \quad \text{and} \quad \models \alpha$$

will have the meanings 'α informs' and 'α is informed', respectively. While $\models$ has a general (universal) meaning of Informing, it can be also semantically particularized in arbitrarily different ways, for instance, in respect to the so-called informational cycle, parallel-serial Informing, or to an arbitrarily distinct application. ◘

DF2 The sign $\models$ can represent one or several cycles of Informing, but can as well be a part of cyclical Informing. A cycle and also the so-called subcycles (substeps) of an informational cycle can be distinguished through the use of the sign $\vdash$. Thus,

$$\alpha \vdash \quad \text{and} \quad \vdash \alpha$$

have the meanings 'α informs in one cycle' and 'α is informed in one cycle'. In this sense, $\models$ and $\vdash$ are operators of Informing through one or several cycles and of Informing in only one single informational cycle, respectively. Since the notion of the verb 'inform' is very broad and embraces all possible linguistic (predicative) expressiveness, the existing and the potential one, the signs $\models$ and $\vdash$ really do represent the most general operators of informational arising, that is of Informing. ◘

Operators $\models$ and $\vdash$ are expressed in the so-called left-right notation, where the left operand informs and the right one is being informed. In a similar manner, the reverse (right-left) operators $\dashv$ and $\dashv$ can be defined, where the informed operand stands on the left side and the informing operand at the right side of the operator. The notions of left-right and right-left operators of arising can be extremely useful within the philosophy of information, Informing, cycling, parallelism, and recurrence. It is possible to combine both sorts of operators, where one of them (for instance, the left-right one) is dominant and the other (for instance, the right-left one) is subordinated to some degree or subjected regarding the first one. Thus, the property of operator $\models$, representing a case of general Informing, can be denoted by $\models^{\dashv}$, for example, where $\models$ in the denotation is dominant to $\dashv$, so that $\models \Delta \dashv$ is the dominance relation between these operators. In this notation, as the upper index of operator $\models$, the sign $\dashv$ signifies the possible influence of the right operand to the left one within a case of general Informing.

2/3

DF3 When information arises from information, the primordial notations $\alpha \models$, $\models \alpha$, and the functional notation $\alpha(\alpha)$ [6] can be replaced by the notation

$$\alpha \models \alpha$$ ◘

DF4 This notation can be used in the case of the recurrent arising instead of the notation

$$\alpha_1 \models \alpha_2$$

in which it is explicitly exposed that α_2 has arisen and arises from α_1 through Informing of α_1 and information α_1 arises also (to some degree) through the backward Informing of α_2. ◘

The notation $\alpha \models \alpha$ is read in the following way: α on the right of the sign $\models$ arises from α on the left of this sign through Informing of α and vice versa; in this mechanism also the arising of α on the left of the sign $\models$ is coming into existence. Within this fact lies the reasonableness of the recurrent notation $\alpha \models \alpha$. The philosophical meaning of $\alpha \models \alpha$ would be that information α is interpreting itself, or that α constitutes its own interpreting and developing (arising) system.

DF5 In general, information α arises out of itself and out of other information α_o; in this case the reasonable notation is

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

Information α_o is understood to be, for instance, an outside information, which remains unaffected by information α. ◘

EX1 Let us look at the following example, and suppose that μ denotes the so-called being's metaphysics (the total information of a being) and σ the so-called sensory information being perceived by the being (which is not the case of informational noise). Then the following notation is senseful:

$$\mu, \sigma \models \mu$$

This is a theoretical case since σ is independent from μ (being not particularly filtered and modulated by μ). In this case, metaphysics μ is developing by itself and by the sensory information σ, thus, μ is recurrent. This case shows how, for instance, the term sensory information depends on the point of observation. If σ is considered as information entering the living sensors and not as information which is already on the way to cortices, for instance, then σ remains unaffected by μ. Otherwise, σ is affected by μ through μ's filtering, modulation, distortion (world model), etc. ◘

2/4

DF6 Certainly, it is possible to treat the arising of information as a recurrent phenomenon also in a much more complex manner. Commonly, two informational cases α and α_1 can interfere mutually. In this case, the meaning of the notation

$$\alpha, \alpha_1 \models \alpha, \alpha_1$$

is that α as well as α_1 are developing from α and α_1, or, that in this process α and α_1, each in its own way, arise recurrently dependent from themselves and from each other. ◘

DF7 Evidently, the notation using the operator of arising $\models$ can be as complex as possible. Generally, it seems quite reasonable to introduce the case

$$\alpha_1, \alpha_2, \dots, \alpha_j \models \alpha_{k+1}, \alpha_{k+2}, \dots, \alpha_{k+m}$$

where j, k, and m are integers. ◘

3. Some Operators within the Cycle of Arising

3/0

According to information principles [4], information arises cyclically and spontaneously. Informational cycle will be understood as a unit or as a step of arising. However, this unit will be divisible into several elementary steps, as they are, for instance, Informing in a narrower (but also broadened) sense, counter-Informing, embedding, etc. In some parts of our discourse, the cycle of informational arising will represent the most elementary scheme of informational arising. For this cycle the symbol $\vdash$ will be introduced. In this sense, symbol $\models$ will represent at least one cycle, however in general an arbitrary number of sequential cycles of arising. However, within an informational cycle (marked by $\vdash$) also a general Informing (marked by $\models$) can occur.

DF8 It is to stress that besides of the sequential nature of the cycle of Informing, parallel components of Informing within the cycle can exist or can come into existence. We shall introduce the parallelness of information into our discourse already in this section. It is to understand that within the meaning (semantics) of the symbols $\models$ and $\vdash$ also a parallel arising can appear, and this can be particularly true in the case $\alpha_1, \alpha_2 \models \alpha_1, \alpha_2$. Now, it is possible to denote several cases of parallel cycles

$$\vdash_1, \vdash_2, \dots, \vdash_n$$

by explicitly parallel operators $\Vdash_1, \Vdash_2, \dots, \Vdash_n$ or simply by the integral parallel cyclic operator $\Vdash$. It means that it will be possible to decompose a cycle $\vdash$ into $\Vdash_n$ or into $\vdash_1, \vdash_2, \dots, \vdash_n$, respectively. ◘

3/1

Processes, arising within the cycle of informational arising, were described in [6]. Several kinds of approximations of an informational cycle have been discussed (for instance, the first, the second, and the third approximation in functional, differential, and integral notation). As it could be comprehended, the cycle of arising can be subdivided into several phases (depending on a particular approximation concept). Particular substeps of the arising cycle can be denoted as follows:

$\vdash_I$ [Informing in the narrower sense],
$\vdash_C$ [counter-Informing],
$\vdash_E$ [informational embedding],

etc. To point out the parallelness of operators within the informational cycle, it can be introduced

$\Vdash_I$ [Informing in parallel],
$\Vdash_C$ [counter-Informing in parallel]
$\Vdash_E$ [informational embedding in parallel],

etc., with other processes appearing in the cycle. On the basis of introduced serial and parallel operators of the type $\vdash_x$ and $\Vdash_x$, respectively, it is possible to express approximations discussed in [6] by the new symbolism, concerning sections 6/4 (functional approach), 7/3 (functional-differential approach), 8/3, 8/4, and 8/5 (functional-differential-integral approach).

3/2

Let us now rewrite the so-called first functional approximation proceeding from information α and closing the informational cycle by putting α_1 equal to α:

S1
$$I = \alpha(\alpha);$$
$$\beta = I(\alpha);$$
$$E = \alpha(\alpha, \beta);$$
$$\alpha = E(\alpha, \beta);$$ ◘

In this cycle, α, I, β, and E denote information, Informing, counter-Information, and embedding, respectively. This functional presentation has the advantage that it can be

understood serially or parallel or that it is a sequence of four expressions or an array of four parallel acting processes.

Applying the subcyclical serial operator $\vdash_x$, one can imagine the following equivalent notation of the four upper equations:

S2
$$\begin{aligned} \alpha &\vdash_\alpha I \\ \alpha &\vdash_I \beta \\ \alpha, \beta &\vdash_\alpha E \\ \alpha, \beta &\vdash_E \alpha \end{aligned}$$ ■

These four "derivations" stand for $\alpha \vdash \alpha$, so the following definition of the informational cycle can be proposed:

DF9 $\alpha \vdash \alpha \quad =_{Df} \quad (\alpha \vdash_\alpha I) \wedge (\alpha \vdash_I \alpha) \wedge (\alpha, \beta \vdash_\alpha E) \wedge (\alpha, \beta \vdash_E \alpha)$ ■

To point out the possibility of the parallelness of an informational cycle, the notation

DF10 $\alpha \Vdash \alpha$ ■

can be used. For pointing out the parallelness of the substeps of a cycle, also the system of four parallel "derivations" can be used:

S3
$$\begin{aligned} \alpha &\Vdash_\alpha I \\ \alpha &\Vdash_I \beta \\ \alpha, \beta &\Vdash_\alpha E \\ \alpha, \beta &\Vdash_E \alpha \end{aligned}$$ ■

The first functional approximation of the informational cycle can be (semantically) captured by the following global definition:

DF11 $\alpha \vdash \alpha \quad =_{Df} \quad \alpha, \beta \vdash_{\alpha,I,E} \alpha, I, \beta, E$ ■

This definition can be logically explained through S2 and S3, which follow S1. Semantics of this definition is not expressing the details of S2 and S3, but how the complexness of components α, I, β and E within α is coming into existence.

3/3

Let us look now at the second functional approximation of the cycle of Informing as described in [6], section 6/4, closing the cycle by $\alpha = \alpha_1$:

S4
$$\begin{aligned} I &= \alpha(\alpha); \\ \beta &= I(\alpha, I); \\ E &= \alpha(\alpha, I, \beta); \\ \alpha &= E(\alpha, \beta, E); \end{aligned}$$ ■

As usual, in this cycle, α, I, β, and E denote information, Informing, counter-information, and embedding, respectively. Instead of S2, for this system the following cyclical notation is obtained:

S5
$$\begin{aligned} \alpha &\vdash_\alpha I \\ \alpha, I &\vdash_I \beta \\ \alpha, I, \beta &\vdash_\alpha E \\ \alpha, \beta, E &\vdash_E \alpha \end{aligned}$$ ■

In respect to S2, in this system, entities β, E, and α depend additionally on I, β, and E, respectively. The dependence of the appearing of informational components became more complex with exception of the component I, which arose from α by α, independently of other cyclic components (I, β, and E). In the similar manner the so-called parallel notation can be constructed. Processes of S5 can be understood as serial, as well as parallel, or, in the best case, as a mixture of serial and parallel information processing. Thus, the definition of the second functional approximation becomes

DF12 $\alpha \vdash \alpha \quad =_{Df} \quad (\alpha \vdash_\alpha I) \wedge (\alpha, I \vdash_I \beta) \wedge (\alpha, I, \beta \vdash_\alpha E) \wedge \alpha, \beta, E \vdash_E \alpha)$ ■

Semantically condensed definition of the informational cycle (in comparison to DF11) becomes

DF13 $\alpha \vdash \alpha \quad =_{Df} \quad \alpha, I, \beta \vdash_{\alpha,I,E} \alpha, I, \beta, E$ ■

This definition has its logical explanation in DF12 or through the system S5.

3/4

It becomes more and more evident that functional approximations of the cycle of Informing are in good correspondence with the so-called derivative notation of informational arising using indexed operators of the type $\Vdash$, $\vdash$, $\nVdash$, etc. Cosequently, functional prescriptions can be symbolically captured into the corresponding indexed derivative operators. In addition, through derivative operators the serialness and parallelness can be expressed explicitly and the constructive nature of informational arising can be expressed more in detail.

The third functional approximation of the cycle of Informing in [6], section 6/4, using again the closure property $\alpha = \alpha_1$, was

S6
$$\begin{aligned} I &= \alpha \circ I(\alpha, I); \\ \beta &= I \circ \beta(\alpha, I, \beta); \\ E &= \alpha \circ E(\alpha, I, \beta, E); \\ \alpha &= E \circ \alpha(\alpha, \beta, E); \end{aligned}$$ ■

In this system, '∘' is used to denote a specific informational composition of informational components. This informational composition depends on particular informational cases and, in the upper system, does not represent an informationally uniform composition. For that reason, it would be much more appropriate to rewrite the system S6 into a new form

S7
$$I = \alpha \circ_1 I(\alpha, I);$$
$$\beta = I \circ_2 \beta(\alpha, I, \beta);$$
$$E = \alpha \circ_3 E(\alpha, I, \beta, E);$$
$$\alpha = E \circ_4 \alpha(\alpha, \beta, E);$$ ◘

It is to understand that '$\circ_1$', '$\circ_2$', '$\circ_3$', and '$\circ_4$' are specific and mutually different compositions of informational components. These compositions by themselves represent further informational components which could be expressed by adequate metainformational functions. Thus, $\alpha \circ \beta$ could be represented, for instance, by the informational (in fact, metainformational) function $\omega(\alpha, \beta)$. Instead of $E = \alpha \circ_3 E(\alpha, I, \beta, E)$, for example, the functional metanotation could be

$$E = \omega_3(\alpha, E)(\alpha, I, \beta, E);$$

The system S6 can now be put into the informationally derivative notation, for instance, in the following way:

S6'
$$\alpha, I \Vdash_{\alpha \circ I} I$$
$$\alpha, I, \beta \Vdash_{I \circ \beta} \beta$$
$$\alpha, I, \beta, E \Vdash_{\alpha \circ E} E$$
$$\alpha, \beta, E \Vdash_{E \circ \alpha} \alpha$$ ◘

In fact, this system represents the definition of the so-called informational cycle which, in a general form, was denoted by $\alpha \models \alpha$ or more specifically by $\alpha \vdash \alpha$. The sign $\Vdash$ in S6' is used for marking the parallel-serial dependence and connectedness of information, irrespective of its arising (as informational object or as informational subject). In a logical notation, the markers $\models$ and $\wedge_{\parallel}$ could be used instead of $\Vdash$. However, since S6' describes Informing within an informational cycle, the so-called cyclical parallel-serial derivation markers of the type $\vdash$ can be particularized, for example, and thus, expressions of S6' can be split into more elementary and adequate form:

S6"
$$\alpha, I \vdash_{\alpha} I; \qquad \alpha, I \vdash_{I} I;$$
$$\alpha, I, \beta \vdash_{I} \beta; \qquad \alpha, I, \beta \vdash_{\beta} \beta;$$
$$\alpha, I, \beta, E \vdash_{\alpha} E; \qquad \alpha, I, \beta, E \vdash_{E} E;$$
$$\alpha, \beta, E \vdash_{E} \alpha; \qquad \alpha, \beta, E \vdash_{\alpha} \alpha;$$ ◘

Resolving the left sides (operands) of the upper expressions, the following four groups of elementary parallel expressions are obtained:

S6^3

(1)
$$\alpha \vdash_{\alpha} I; \quad I \vdash_{\alpha} I; \quad \alpha \vdash_{I} I; \quad I \vdash_{I} I;$$

(2)
$$\alpha \vdash_{I} \beta; \quad I \vdash_{I} \beta; \quad \beta \vdash_{I} \beta;$$
$$\alpha \vdash_{\beta} \beta; \quad I \vdash_{\beta} \beta; \quad \beta \vdash_{\beta} \beta;$$

(3)
$$\alpha \vdash_{\alpha} E; \quad I \vdash_{\alpha} E; \quad \beta \vdash_{\alpha} E; \quad E \vdash_{\alpha} E;$$
$$\alpha \vdash_{E} E; \quad I \vdash_{E} E; \quad \beta \vdash_{E} E; \quad E \vdash_{E} E;$$

(4)
$$\alpha \vdash_{E} \alpha; \quad \beta \vdash_{E} \alpha; \quad E \vdash_{E} \alpha;$$
$$\alpha \vdash_{\alpha} \alpha; \quad \beta \vdash_{\alpha} \alpha; \quad E \vdash_{\alpha} \alpha;$$ ◘

These informational processes are parallel and show informational complexity of a case of informational cycle.

4. Other Possible Interpretations of the Informational Cycle

4/0

In the previous section, the informational cycle was interpreted through the functional concept ([6], 6/4) and through the indexed (particularized) operators of arising within the informational cycle (cycle of Informing). Within this cycle several cases of information and of Informing have been introduced such as source information, counter-information, information of embedding, information of Informing, etc., as well as informational processes which actively generate and change information such as Informing, counter-Informing, embedding, etc. In this sense, we dealt with some kinds of basic cyclic informational forms and processes.

In [6], section 7/3, another interpretation of the informational cycle has been given, introducing the notion of informational differentiation, which concerns a particular cyclic information, called counter-information. Counter-information β was obtained through the process of informational differentiation of a dynamic form of source information, marked by $\alpha(\alpha)$, by information α. Symbolically, this process was labeled by

$$\beta = D_{\alpha}\, \alpha(\alpha)$$

where D_{α} denotes the operation of differentiation (instead of $d/d\alpha$). In this case, D can be understood as a modal operator of differentiation where α is the agent of differentiation. It can be fixed that this process of informational differentiation represents the so-called counter-Informing. It is to understand that counter-Informing arises in parallel with counter-information and that they have influence upon each other.

It is evident that D has the function of a classical modal operator. The notation $D_x\, y$ can be read as 'agent x derives (or differentiates) y'. Instead of D, which is a particular modal operator concerning counter-Informing, the notation $C_{\alpha}\, \alpha_1$ can be used, which is read as 'information α counter-informs information α_1'. Now, the result of this counter-Informing can be labeled as β, which represents the arisen counter-information, so that a transformation, using the symbolical equality

$$\beta = C_I\, \alpha$$

can be introduced meaningfully.

Let us rewrite the so-called third approximative system ([6], 7/3), putting $\alpha = \alpha_1$, in the form

S8
$$I = \alpha \circ I(\alpha, I);$$
$$\beta = I \circ (C_{\alpha}\alpha(\alpha) \diamond_1 C_I\alpha(\alpha) \diamond_2 C_{\beta}\alpha(\alpha) \diamond_3 C_{\alpha}I \diamond_4 C_{\beta}I \diamond_5 C_{\alpha}\beta \diamond_6 C_I\beta);$$
$$E = \alpha \circ E(\alpha, I, \beta, E);$$
$$\alpha = E \circ \alpha(\alpha, \beta, E);$$ ◘

This system can be rewritten into

S8' $\alpha, I \vdash_{\alpha \circ I} I;$
$C_\alpha\alpha, C_\alpha I, C_\alpha\beta, C_I\alpha, C_I\beta, C_\beta\alpha, C_\beta I \vdash_I \beta;$
$\alpha, I, \beta, E \vdash_{\alpha \circ E} E;$
$\alpha, \beta, E \vdash_{E \circ \alpha} \alpha;$ ◘

Similarly as in the case of S6", the system S8' can be split into detailed parallel processes too.

The last interpretation of the informational cycle will be the counter-Informing/embedding one which proceeds from Section 8 in [6]. The consequence of embedding E in an informational cycle is the appearing of the so-called embedding information ε which connects arisen counter-information β into source information α. In this case, operation of embedding E becomes a modal operation of various informational agents. The modal operator of embedding performs by the production of information of embedding ε which is necessary for integration of counter-information β into source information α (by means of information ε). For an integrational modal operator INT two of its agents have to be considered: the informational domain into which counter-information β will be integrated (the first informational agent) and information by means of which the act of integration will be performed (the second informational agent). In this way, the expression

$$\varepsilon = INT_{\alpha,\beta}\ I$$

has the following meaning: counter-information β integrates Informing I into informational domain α, producing information of integrative connectivity ε.

For the second approximative system from Section 8/5 [6] we have at $\alpha = \alpha_1$:

S9 $I = \alpha(\alpha);$
$\beta = C_\alpha\alpha(\alpha) \diamond C_I\alpha(\alpha);$
$\varepsilon = INT_{\alpha,\alpha}\beta \bullet INT_{\alpha,\beta}I;$
$\alpha = \alpha \circ \varepsilon(\alpha, \beta, \varepsilon) =$
$INT_{\alpha,\beta}\varepsilon \diamond_1 INT_{\varepsilon,\varepsilon}\beta \diamond_2 INT_{\beta,\alpha}\varepsilon;$ ◘

This system can be rewritten into a composite parallel-serial form, for instance:

S9' $\alpha \Vdash_\alpha I;$
$C_\alpha\alpha, C_I\alpha \Vdash \beta;$
$INT_{\alpha,\alpha}\beta, INT_{\alpha,\beta}I \Vdash \varepsilon;$
$INT_{\alpha,\beta}e, INT_{\varepsilon,\varepsilon}\beta, INT_{\beta,\alpha}\varepsilon \Vdash \alpha;$ ◘

This system can be split into more detailed parallel informational processes using particularized parallel operators of the type $\Vdash$.

5. Parallel Informational Processes

Currently, parallelism belongs to the most concealed and unrevealed informational phenomena which calls for philosophical and technological illumination. Parallel informational processes open the most complex spatial and temporal interweaving, dependence, and arising as have ever been imaginable. Although, human mind in its basic structure and organization operates in a parallel and parallel-serial manner, on the the higher cortical levels, in its global informational organization (e.g., in functions of awareness, reasoning, and intention) it appears, behaves, and experiences primarily as a serial or sequential apparatus. Since the basic parallel-serial informational structure and organization of the mind are not directly reflected in human awareness and conscious reasoning, the conscious part of human mind does not dispose of the required fundamental experience for an adequate conceptualization of informational parallelness. Thus, a philosophical background of informational parallelness has to be investigated and constructed, for it cannot be discovered sufficiently in detail.

Two kinds of parallel informational operators will be used: the general ($\Vdash$) and the cyclical ones ($\Vdash$). General and cyclical parallel operators of Informing can be particularized (through their indexing or replacing by already known, however, semantically broadened ones) or universalized (complexed). The consequence of introducing parallel operators of Informing can be the arising of specific theorems which govern the concept and construction of parallel structured and organized information systems.

DF14 The general parallel operator of Informing will be denoted by $\Vdash$. Thus, for

$$\alpha \Vdash \quad \text{and} \quad \Vdash \alpha$$

the meaning will be 'α informs in parallel to (some other parallel processes)' and 'α is informed in parallel to (some other parallel processes)'. Again, $\Vdash$ has a universal meaning of parallel Informing and it can be particularized arbitrarily (through its indexing) or universalized. ◘

DF15 Within an informational cycle, it is possible to introduce (the already discussed) parallel cyclical operators of the type $\Vdash$. Thus,

$$\alpha \Vdash \quad \text{and} \quad \Vdash \alpha$$

have the meaning 'α informs in a cycle in parallel to (some other parallel cyclical processes)' and 'α is informed in a cycle in parallel to (some other parallel cyclical processes)'. ◘

DF16 Similarly as in definitions DF3 and DF4, it is possible to define, for instance,

$$\alpha \Vdash \alpha \quad \text{and} \quad \alpha_1 \Vdash \alpha_2$$

where the operator $\Vdash$ in both cases guarantees the existence of at least one parallel process of the form $\alpha_3 \Vdash \alpha_4$.

Introducing parallel processes, some generall cases of Informing can be determined in more detail. For instance, the case of DF5 can be defined in the following way:

S10 $\quad \alpha, \alpha_o \models c \;=_{Df}\; \alpha \Vdash \alpha, \alpha_o \Vdash \alpha$ ▣

It is evident that $\alpha, \alpha_o \Vdash \alpha$ concerns the parallelness of processing through or within the operator $\models$, however, in the parallel system $\alpha \Vdash \alpha, \alpha_o \Vdash \alpha$ this parallelness is *more* explicit, although it is not determined in a greater detail. In this respect, a parallel system using parallel operators has to be and can be further formally decomposed by introduction of additional parallel operators, enabling also to explicate the communicational (or coordinational) connectivity of parallel information.

Similarly, DF6 can be redefined as the following parallel system:

S11 $\quad \alpha \Vdash \alpha, \alpha \Vdash \alpha_1, \alpha_1 \Vdash \alpha, \alpha_1 \Vdash \alpha_1$ ▣

As mentioned, parallel connections and interactions among occurring informational entities α and α_1 of S11 can exist in imaginably possible way. The question is if *such connections and interactions are possible* also on the level of occurring parallel operators $\Vdash$. Untill now, we have not studied informational decomposition of an informational operator, so that an informational inference in and out of its body would be possible.

As it is evident, DF7 can be rewritten into its parallel structured equivalent in the following manner:

S12 $\quad \alpha_1 \Vdash \alpha_{k+1}, \alpha_1 \Vdash \alpha_{k+2}, \cdots, \alpha_1 \Vdash \alpha_{k+m},$
$\alpha_2 \Vdash \alpha_{k+1}, \alpha_2 \Vdash \alpha_{k+2}, \cdots, \alpha_2 \Vdash \alpha_{k+m},$
. . . . . .
$\alpha_j \Vdash \alpha_{k+1}, \alpha_j \Vdash \alpha_{k+2}, \cdots, \alpha_j \Vdash \alpha_{k+m}$ ▣

Now, further parallel decomposition of this system can take its course. It becomes quite obvious that additional parallel informational operators must be introduced to express the possibilities of interprocessing communication and interaction.

6. Particularization and Universalization of Informational Operators

In the previous discussion and examples it was shown how a logical system can proceed from a sufficiently universal informational operator from which every particular case of operation can be constructed. Vice versa, a given operator can be universalized to symbolize even a more complex function. A universalization of an operator can be obtained through its definition in a recurrent manner.

In a general case, Informing I_α of information α arises through an informational process

$$\alpha \models I_\alpha$$

This formula explicitly expresses how α informs, namely through its Informing I_α, which is exactly the way in which α informs at all. Let us look at the following six possibilities:

$$\alpha \models I_\alpha \equiv \alpha \models^{\dashv} I_\alpha \vee \alpha \models^{\not\dashv} I_\alpha \vee \alpha \not\models^{\dashv} I_\alpha \vee \alpha \dashv^{\models} I_\alpha \vee \alpha \dashv^{\not\models} I_\alpha \vee \alpha \not\dashv^{\models} I_\alpha$$

These cases can be explained in the following way:

(1) $\quad \alpha \models^{\dashv} I_\alpha \quad$ with $\quad \models \Delta \dashv$

says that α is dominantly informing I_α and that a non-dominant Informing of I_α on α exists. Logically, this can be explained by

$$\alpha \models^{\dashv} I_\alpha \equiv ((\exists(\alpha \models I_\alpha)(\models \Delta \dashv)) \wedge_{\|} (\exists(\alpha \dashv I_\alpha)(P(\dashv \Delta \models))))$$

This case corresponds adequately to the idea of informational arising or Informing of information. On the right side of the equivalence sign, the meaning of the sign $\models$ is the simple one (not recurrent).

(2) $\quad \alpha \dashv^{\models} I_\alpha \quad$ with $\quad \dashv \Delta \models$

says that I_α is dominantly informing α and that a *non-dominant Informing of* α *on* I_α exists. Logically, this can be described by

$$\alpha \dashv^{\models} I_\alpha \equiv ((\exists(\alpha \dashv I_\alpha)(\dashv \Delta \models)) \wedge_{\|} (\exists(\alpha \models I_\alpha)(\neg(\models \Delta \dashv))))$$

In this case, I_α is influencing α to a higher extent than it is the case vice versa. Informing of information is stronger, more effective than is information by itself, from which this Informing arises. For instance, questioning which was generated through a given question, can become more relevant from the initial question itself. This case is, within a normal idea about informational arising or Informing of information, unrevealed, and to some extent unusual, although it can occur accidently.

(3) $\quad \alpha \models^{\not\dashv} I_\alpha \quad$ with $\quad \models \Delta \not\dashv$

This case is dilemmatic because of $\models \Delta \not\dashv$. The negation of the relation $\dashv$ (this is the upper relation $\not\dashv$) means that Informing in the backward direction is not taking place. The question is, if $\models$ and $\not\dashv$ can be put into relation of dominance at all. Does it mean that in the relation $\alpha \not\dashv I_\alpha$ the influence of I_α on α is hidden, in some way? This influence of $\not\dashv$ can be understood in the manner that α is informed about the non-influence of I_α on α. Formally, this can be described again with

$$\alpha \models^{\not\dashv} I_\alpha \equiv ((\exists(\alpha \models I_\alpha)(\models \Delta \not\dashv)) \wedge_{\|} (\exists(\alpha \not\dashv I_\alpha)(\neg(\not\dashv \Delta \models))))$$

In this case, Informing I_α arises from α, where I_α does not influence α at all.

(4) $\alpha \dashv^{\models} I_{\alpha}$ with $\dashv \Delta \models$

is similar to the case (1), but with exchanged operators.

(5) $\alpha \dashv^{\not\models} I_{\alpha}$ with $\dashv \Delta \not\models$

is similar to the case (4), but with exchanged operators.

(6) $\alpha \not\dashv^{\models} I_{\alpha}$ with $\not\dashv \Delta \models$

is similar to the case (2), but with exchanged operators.

Up to now we have discussed the following operators of the type $\models$: operators concerning general, general cyclical, parallel cyclical, and general parallel Informing or processes of Informing. In this course we can have the following four sets of definitions:

DF17 This definition concerns the general type of informational operators:

$\models$ is the general, main, usually from the left to the right directed logical operator of Informing which can seize a particular or universal operation of Informing; it can be generally a multiplex (down to unary) operator, of the infix, prefix, or postfix type; it can be arbitrarily particularized or universalized; in an expression (informational well-formed formula) it can function in one direction (e.g. left-right) or another (right-left), or in both directions, where directional operations can be in a defined (functional) relation (e.g., of dominance, where the operation in one direction is subordinated in concern to the operation in the reverse direction). Usually, operator $\models$ can represent any other operator including all of the listed informational operators which follow;

$\dashv$ is similar to $\models$, but with the opposite (right-left) direction of operation. In this case the dominant direction is from the right to the left side of operator (if it has its function in the opposite direction at all). Informational operator $\dashv$ can be useful in cases dealing with parallel informational processes, so that two of them can be linearly presented (composed) within a single formula using general informational operators.

$\not\models$ is operational, informational negation of the general operator of Informing $\models$. The meaning of this operator is, for instance that Informing of the type $\models$ in the left-right direction does not exist or does not arise;

$\not\dashv$ is similar to $\not\models$, but with the opposite (right-left) direction of operation. ◘

DF18 This definition concerns the general cyclical type of informational operators. The difference between a general and a general cyclical type of operators is that an informational cycle concerns a particular unit of general Informing within which this unit has a typical, the so-called cyclical Informing (e.g. counter-Informing, embedding of information, etc.). In this regard, general cyclical operators of Informing are similar to the general ones in DF17. Thus,

$\vdash$ is the general cyclical operator which governs the Informing within the informational cycle. This operator can represent serial or parallel processing and any particular operation in the dominantly left-right direction within a cycle;

$\dashv$ is similar to $\vdash$, but with the opposite (right-left) direction of operation;

$\nvdash$ is operational, informational negation of the general cyclical operator of Informing $\vdash$. The meaning of this operator is that Informing of the type $\vdash$ in the left-right direction does not exist or does not arise;

$\not\dashv$ is similar to $\nvdash$, but with the opposite (dominantly right-left) direction of operation. ◘

DF19 This definition concerns the parallel cyclical type of informational operators. Parallel cyclical operators appear only within an informational cycle. The difference between a parallel and a non-parallel operator is that parallel cyclical informational operators appearing in an informational system (of informational processes) denote the parallel connectedness of processes of the system to which they belong. This connectedness has the meaning of mutual communication (and from this communication proceeding co-ordination of involved parallel processes) in a system. This principle of connectedness is valid also for differently particularized or univarsalized informational operators. The appearance of a parallel operator means that in the informational system at least one other parallel process exists which is in a certain informational relation with the source process, but may also be in relation with other processes within the system. Again, the following four cases are relevant:

$\Vdash$ is the parallel cyclical operator of Informing and designates that at least one parallel cyclical operator exists in a parallel system. The meaning of this operator can be similar to the meanings of operators $\models$ and $\vdash$, with the exception that $\Vdash$ represents parallelism within an informational cycle.

$\dashv\!\!\dashv$ is the case of the opposite (from the right to the left) direction of $\Vdash$.

$\nVdash$ is the operational, informational negation of operator $\Vdash$. The meaning of this operator is that within the cycle there cannot exist an informational process parallel to the process which it concerns.

$\not\dashv\!\!\dashv$ is similar to $\nVdash$, however points in the opposite direction. ◘

DF 20 The last group of informational operators are the parallel general ones. These operators can be used within or outside the concept of the cycle, that is for cyclical or general purposes. The meaning of these operators concerns their generallity and parallelness simultaneously. Again, four cases of parallel general operators can be significant:

$\models\!\!\!\models$ is the most general, parallel logical operator of Informing which can be particularized and universalized as already described in the previous cases of operators $\models$, $\vdash$, and $\models\!\!\!\vdash$. Parallelness belongs to the most relevant, but also unrevealed concepts of information. In this respect, additional meaning (semantics) of parallel consequences of such a general operator can be studied.

$\dashv\!\!\!\models$ is the opposite case of $\models\!\!\!\models$. The notation using this operator enables (as already mentioned) a more compact or dense description even in cases of parallel informational processes.

$\models\!\!\!\not\models$ is operationally expressed informational negation of the possibility that to the (parallel) process possessing operator $\models\!\!\!\not\models$ another parallel process possessing operator $\models\!\!\!\models$ could exist. However, there can exist parallel processes to the process possessing $\models\!\!\!\not\models$ which include other parallel informational operators (e.g., $\dashv\!\!\!\models$, $\not\dashv\!\!\!\models$, $\models\!\!\!\vdash$, $\dashv\!\!\!\vdash$, etc.).

$\not\dashv\!\!\!\models$ has similar meaning as $\models\!\!\!\not\models$, but is acting in the opposite direction. □

At the formation of informational formulae, all of the the discussed operational operators can be adapted accordingly to arising particular and universal needs and goals of the describing informational problems, scenarios, etc.

7. Some Possible Informational Scenarios of Questioning

In everyday conversation, the most general case of monologue, dialogue, or multilogue is an informational scenario of questioning. In a process of speech, talk, or discussion, the principle of questioning seems to be the most natural one and is the real motor of development of discourse and of the arising of living, beings-concerning information, coming into existence during a conversation.

What belongs to questioning which is understood as to be an informational phenomenon? At the beginning there is a question which arises or has been already arisen within a developing information. The appearance of a question causes and triggers the process of questioning concerning the question. Within this process answers arise which are embedded into surrounding information and regard the subject of questioning. The arisen answers give rise to the appearance of additional questions or change and complete the original ones. Thus, a further questioning comes into existence which can deliver new answers again, etc. The process of questioning can continue in this or another way until its Informing becomes exhausted within its arising of possible contents, aims, or reasons of questioning. Finally, this process stops, so that Informing of information can continue its unforeseeable course of arising.

The question which appears now is the following: How can the described course of questioning be expressed in an informationally formalistic way? Is it at all possible to express it adequately and to what degree of logical relevance does it extend? Informational formalization of the process of questioning can certainly deliver constructive concepts for an implementation of questioning, for instance, in an intelligible expert system. On the other hand, the process of questioning represents a sufficiently general case of Informing, so that it can be exemplarily used in the cases of a conctructive conceptualization of similar informational problems.

SC1 There are several scenarios of the process of questioning possible. Let first us explain the following one:

$$
\begin{aligned}
(\models \alpha) \vdash\ & (\models \alpha \dashv q \lrcorner) \vdash \\
& ((\models \alpha \dashv q) =_{Df} (q \models \alpha)) \vdash \\
& ((q \models \alpha \models a) \wedge (a \models q)) \vdash \\
& ((q,\ a,\ \alpha \models q,\ a,\ \alpha) \urcorner)
\end{aligned}
$$

As it can be recognized from the last expression, informational process which uses informational cycles (the operator $\vdash$) was chosen. Within distinct informational cycles, the operators $\models$ for general Informing are used. Of course, within distinct informational cycles arbitrary operations can be performed, including the operation of definitional equivalence and other logical operations. In this respect, the last expression is absolutely logically compact. Let us look now at the meaning, or better at the interpretation of the last expression.

At the beginning, information α is informed. This is the cycle in which information α is in the state to be informed or to be open for receiving of information. In the next cycle, a question q appears (the operator $\lrcorner$). It is not explicitly expressed where is the question q arriving from, whether from information α or from information which surrounds α. The activity of Informing of α now takes over the question q, which becomes evident in the next informational cycle using definitional equivalence for the description of this fact. However, 'q informs α' has the meaning that α influences q through the hidden feedback operator of Informing $\dashv$ in $\models$. This phenomenon of Informing is explicated in the next cycle where q informs α, α informs the arisen answer a, and a informs backward q again, etc. In the last cycle, the process of questioning stops (the operator $\urcorner$) where the meaning of the last part of the expression is as follows:

$$
\begin{aligned}
((q,\ a,\ \alpha \models q,\ a,\ \alpha) \urcorner) \equiv\ & ((q \models q) \wedge (q \models a) \wedge (q \models \alpha) \wedge \\
& (a \models q) \wedge (a \models a) \wedge (a \models \alpha) \wedge \\
& (\alpha \models q) \wedge (\alpha \models a) \wedge (\alpha \models \alpha) \wedge \\
& (q,\ a,\ \alpha \text{ stop to inform mutually})) \quad \square
\end{aligned}
$$

SC2 Another scenario of questioning could be, for instance, the following:

$$
\begin{aligned}
(\alpha \models \alpha) \vdash\ & (\alpha \models \alpha \dashv q \lrcorner) \vdash \\
& \alpha,\ q \models \alpha,\ a) \vdash \\
& (\alpha,\ q,\ a \models \alpha,\ a) \vdash \\
& (\alpha,\ q,\ a \models \alpha,\ q,\ a) \vdash \\
& (a \subset_{\varepsilon} \alpha \urcorner)
\end{aligned}
$$

We start from the assumption that α initially informs in itself, that it is in an informationally active waiting state. In this state of self-Informing of α, in the next cycle, a question q begins (the operator $\lrcorner$) to arrive into the field of relevance of α. Information α and the question q now inform mutually in the subsequent cycle to produce an answer a and correspondingly change α. The answer a becomes an additional item of Informing within the context of α and q. Before the end of Informing of this case, source information α, the question q, and the answer a inform mutually dependent when finnally, the answer a is embedded (the operator $\subset_{\varepsilon}$) into source information α, and the scenario stops (the operator $\neg$). ◘

References

[1] H. L. Dreyfus and S. E. Dreyfus: *Mind over Machine.* The Free Press, Macmillan, New York 1986.

[2] A. P. Zeleznikar: *On the Way to Information.* Informatica 11 (1987), No. 1, 4-11.

[3] A. P. Zeleznikar: *Information Determinations I.* Informatica 11 (1987), No. 2, 3-17.

[4] A. P. Zeleznikar: *Principles of Information.* Informatica 11 (1987), No. 3, 9-17.

[5] A. P. Zeleznikar: *Information Determinations II.* Informatica 11 (1987), No. 4, 8-25.

[6] A. P. Zeleznikar: *Problems of Rational Understanding of Information.* Informatica 12 (1988), No. 2, 31-46.

[7] L. A. Steen (Editor): *Mathematics Today.* Twelve Informal Essays. Springer-Verlag. New York (Third Printing, 1984).

AN ANALYSIS OF INFORMATION SYSTEMS DESIGN METHODOLOGIES

INFORMATICA 3/1988

Krista Rizman
Ivan Rozman
Anton Zorman
Tehniška fakulteta Maribor

UDK 681.3.:519.863

ABSTRACT
To improve development and use of information systems methodologies, these have to be discussed and studied from many aspects. We have analysed JSD, ISAC, SA-SD and the Warnier/Orr methodology. The contents of this paper is not a description of studied methodologies. It is the description of our findings and the results of evaluating the practical value of the methodologies in relative terms by comparing them. We characterize the methodologies according to their life cycle coverage, their representation shemes, learnability, their behavior in the real time environment and automated tools by which they are supported. Our main point here lies in demonstrating that each of the four methodologies is relative powerful in some circumstances and for some systems.

1 INTRODUCTION

The use of computers has spread to all areas of labour and life and the way of use has changed. In the first period of use, computers were used only for calculation. Almost all data processings were numerical. The main point of use has moved from calculation to storing and searching data - information. There has been a great progress of new approaches, methodologies and tools for developing of information systems respectivly application by the last decade. The information system is a system with the following tasks: creating, collecting, processing and distributing informations. Information systems could be developed only if they in some way can improve some activities of information process.

The progress of computer technology, especialy the fall of prices, caused the mass use of computers. The development of applications has become a bottleneck. This is a reason why the languages of the fourth generation (application = functional languages: LISP, PROLOG, query languages...) and a great number of computer aided tools for system analysis and design have been produced.

There are a great number of graphic tools and methodologies that can be used for analysing and design of information systems. The problem is to choose one from this set of methodologies, that will be the most efficient, easy for use and will give the best results. There are many questions: 'Which methodology to choose?', 'Which is the best?', ' Which is the most general?', 'What are advantages of each of them?'

Which methodology to use? This is a difficult question because it depends on your needs, on the type, the extension and the complexity of information systems you want to develope, on your experience, style of thinking and probably on your knowledge of principles of different methodologies.

We restrict our study on methodologies that are shown in table 1, mainly because we think they are the most efficient and widespread used for developing information systems. This paper presents the results of study the different methodologies with the purpose to answer some questions above.

Table 1: In the following we give the methodologies covered in this report along with developers

Methodology mnemonic	Full name of methodology	Developers
SDM	System Development Methodology	G.F.Hice, W.S.Turner
SA-SD	Structured Analysis and Structured Design	De Marco Yourdon
ISAC	Information Systems Work and Analysis of Changes	Mats Lundeberg
JSD	Jackson System Development	Michael Jackson
	Warnier/Orr-method	Warnier, Orr

It must be pointed out that our analysis draws on the referenced literature. This is important because methodologies are not finished products. They do not have precise characteristics. They are improved through time. Methodologies in a practical use can be adapted to chainging environments and circumstances. On the other hand, the authors often emphasize ascpects considered as most original, while aspects regarded as more usual are left out.

Several points of view can be used to analyse a methodology and all of them must be considered in a complete analysis.

2 LIFE CYCLE COVERAGE

By analysing the mentioned methodologies, we find out that they have much in common. They have a great diversity in form, in original principles and in name of each phase, but they agree with basic elements that need to be defined. A lot of them cover almost all phases of a life cycle(table 2).

The life cycle is an important concept in discussing methodologies. Our view is that an efficient methodology must support a process of activity that covers the entire life cycle. It does little good to have a methodology for design if there is not a procedure for specifing requirements and functions which are used for the design and the system that must be built. There are numerous life cycle models in use and many of them separate analysis and functional specification activities./PORC83/ We merge both of them into one phase (analysis) because the discussed methodologies do not distinguish between these two steps. For both phases almost all of them (except the ISAC methodology) support the same graphical diagrams and other tools and in both the users are most included.

In this paper each methodology is presented through the description of the following life cycle steps:

- analysis,
- design,
- implementation,
- validation,
- evolution.

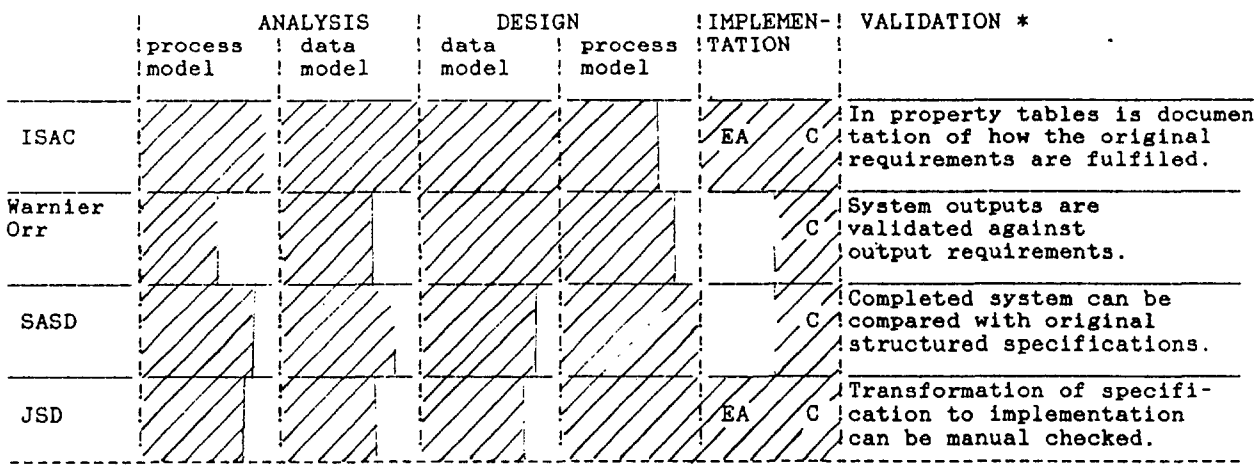
2.1 Analysis

Analysis is the first step of any information systems development activity. The result of this step is besides the requirements definition also a functional specification - description of system functions and an answer to the question : 'What should the system do?'. Functional specifications are used during the design phase as a checkpoint against which to validate the design.
The successful analysis includes communications with the users. The analysis must be able to bound a problem and to identify those areas where the information system is useful and practical.

A particularly effective method of analysis is modeling. Models represent the problem and the real world in a formal form. Models used for analysis are graphical diagrams, graphs and tables.

Because of the complexity of problems and systems, methodologies must support a problem decomposition which can be procedural or data flow or data abstraction.

All discussed methodologies are performed through an analysis of the data, either data structures or data flows! The data orientation is sensible because data are more stable than processes. SA-SD and ISAC analyse data flows, but the Warnier methodology analyses the data



- represents how detail is a particular phase covered

EA - methodology contains a special phase with the purpose to choose specific equipment and then to adapat the equipment independent solution to this choice

c - coding

* - how the completed system is validated against the original requirements

Table 2: Life cycle coverage

structures of the outputs. JSD analyses the entity/action structure of the real world. (Figure 3)

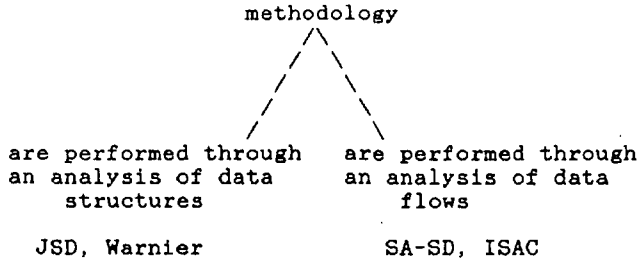


Figure 3: A simple division of the information systems design methodologies

Our view is that the data flow methodologies are more understandable than the methodologies which base on the data structure. A data flow diagram (DFD) is a better tool for understanding and more perfect representation of any information system than data structure diagrams. It represents both flows and actions. (Figure 5 shows an example of DFD.) It is a network diagram that can be easy used for bounding the system. First of all, we draw the diagram with only one action and with input and output data flows. They are then decomposed. We think that so the users and the analysts can easily, systematically and by degrees, with the help of DFD, make the requirement definitions and the functional specifications.

The SA-SD methodology is very useful for working in a team because of the logical functional decomposition and well-known input and output data of any decomposed action.

More complex and detailed analysing process can be done by ISAC. Besides the information (message) flows we can also describe the real flows (persons and objects). The picture of current and future information system is complete.
ISAC is very strong in the early phases of the system life cycle: the change analysis, the activity analysis and the information analysis. The ISAC approach is extensive and comprehensive!
In ISAC interest groups are studied thoroughly and described both with the problems they have. This is a part of the change analysis. The ISAC approach is widespread used in the Scandinavian countries.

The weak point of ISAC is that the graphical notation for the information analysis is redundant because all information contained in information graphs (I-graphs) is derived from activity graphs (A- graphs) from the change or the activity analysis. (Figure 6 shows an example of A-graph!)
The weak point is also that it is necessary to illustrate two identical information sets in the same I-graph because of the hierarchical way of description. Two sets are needed when one information set is output set of a graph and at the same time is a precedent to other set in the same graph.
I-graphs show information sets and precedence relations among information sets, but C-graphs (component relation graphs) describe the contents and the structure of the information set. The methodology does not provide that the same information set will have only one C-graph. It depends on when the particular C-graph is created.

We think that many of definitions and work of the information analyse could be overcome by use more powerful information model such as extended entity-attribute diagram to replace both information graphs and component graphs.

Weakness of JSD is the first step of methodology (entity-action step), by which we analyse data and actions of the real world. It seems to us that in this step the methodology does not provide such a graphical tool which can help the users and the analysts to edit, colect and represent specifications (especialy all entities).
Graphical notations are useful in showing the interrelationships between the system components, which enable easy communications between the users and the analysts and so help both of them to build the complete information system.
JSD does not provide such a graphical tool. Jackson suggests a simple process to make a wide list of entities and actions: nouns which appear in the users description of the real world are identified as potential entities, but verbs as possible actions. The users many times forget to mention some parts of the reality because they do not have resource for sistematic description of often very complex sistems. Unfortunately, the complete list of entities and actions is request and condition for success of entire development. JSD does not support graphical presentations of links between entities of the entire system, from which can the users and the analysts quickly find out the missing entities.

The JSD methodology is little oposite to the other information systems development methodologies. They tend to more exact analysis with purpose to build a better system with less price to meet the needs of its users and to reduce costs of correcting.
The second tendency is reducing of returning to previous steps. Of course, there is an iteration, but we all want to reduce it as much as possible. Modern methodologies and computer aided tools include mechanism to reduce it to minimum.

In the **Warnier methodology** the first step is to determine which are required outputs. The answer is quite obvious when the system is not too big. Else we have to subdivide the real system into several smaller. Many times the subdivision will be done according to the organisation of the firm. Analyst may help formulating questions and so help to create a list of required outputs. The methodology does not provide any graphical tools to help in this first step.

All the methodologies except JSD apply the hierarchical decomposition. Of course, levels of detail are related to complexity and vagueness. The vagueness concerns the early phases of life cycle, when the functional and the data system may be fuzzy and there is no clear idea how the system will work. In this context a crude information analysis is quite good alternative. The possibility of the crude information analysis is embedded in the ISAC methodology. We start to build new system with the crude information analysis in the change analysis and then we end with the detailed analysis in the information analysis phase. The crude information analysis is also supported by SA-SD, which enables simple execution of the functional decomposition.
Our view is that the most detailed analysis is provided by ISAC, then follow SA-SD, JSD and the Warnier methodology.

2.2 **Design**

is the process of determining the architecture

of the system components, the algorithms to be used and the logical data structures. This phase is an answer to the question: 'How will the system perform the functions defined in the previous phase?' . An output of design activities can be used by the programmer to implement the system. We must emphasize that coupling and coheison are the simple judge of whether a design strategy produces good designs or bad designs.

A developer must be able not to continue only forward to the next phase of the life cycle, but also back to a previous phase. The need for this is the fact that work must be changed and any necessary corrections can be made. It is important to note that information lost at a particular phase is generally lost forever with a bad result on the system. For example, if a requirement are not documented, it will not appear in the functional specification and it will cause the failure of the system.

All the methodologies except JSD are very strong based on the levels of abstraction and on the hierarchical decomposition, where there is always the problem of wheter the model at the lower level satisfies the specification fixed at the upper levels. This problem can partly be dispatch by detailed transformation rules from an upper level to a lower one and by automated tools.
SA-SD supports two transformation rules: a transform analysis and an analysis of activities for producing a structure diagram from data flow diagrams. We must tell that structured diagrams can not be made only by the transaction and the transform analysis but it requires some judgement and common sense on the part of the designer.

This strategy is considered in the Warnier methodology well but in ISAC only particular. ISAC makes levels of abstraction quite visible, but there is not a visible connections between the analysis phase and the design phase. Perhaps it is the reason in use of other method (Jackson Structured Programming) for the design.

2.3 Implementation

is the production of executable code. Coding transforms algorithms into functions or procedures and logical data structures into phisical data structures. It must be pointed out that good coding cannot make up for poor analysis or design. Good coding cannot make bad information systems good! This phase is an answer to the question: 'How can we run this system on machine available to us?' .
The ISAC and the JSD methodology enable design which is not confused with implementation. The delimitation between design and implementation is in the ISAC and the JSD methodology very rigorous. Not before the latest phase we include the use of existing hardware and programming languages for realization developed system.
This is an advantage of both methods, because the design system is more transferable and more portable. We can use it with little changes on different hardware because only the last phase must be changed.

The choice of the hardware and the software needed and technical asspects of the implementation the Warnier mathodology and SA-SD do not solve.

2.4 Validation

is the process of determining that a system correctly performs those functions described in the functional specification. It is often a step performed as a part of each phase where we must verify that the phase correctly carries out the intent of the previous step. The validation of code may be done either through testing or formal proof of correctnes.
The methodology must support determination of system correctness through the life cycle. Methodologies usualy enable correspondence between the results of one stage of development and the previous stage.
Table 2 shows how the whole system can be validated against the original requirements.

2.5 Evolution

Systems go through many versions during their lifetimes. The development methodology can help in evolution phase by providing system documentation and, of course, a well structured software system that is easy modified by people making the system changes. The great emphasis to a well structure of a program is given in the SA-SD methodology. The factors contributing to interactions between systems components (modules) and the coheison of individual systems components are very well described. /Your79/ We tent to the greater coheison of individual modules in the system and the lower coupling between modules.

3 INTERMEDIATE WORK PRODUCTS

By methodology we mean a number of coherent work steps including rules for types of documentation that are produced during these work steps. The documentation must be a natural part of work and not something that you do afterward! Table 4 shows the steps of all the four discussed methodologies and the products that are produced at each step! Figures 5,6,7 and 8 show the working procedure of all the four methodologies. Each working procedure is presented by the diagram, which is particularly signicifant for each of the four methodologies.

We have already emhasized that graphical tools in ISAC seem to us redundant because the contents in I- graphs is the same as in A - graphs. But the purpose of using both graphs is different. A- graphs give an overview of the connections between the information system and its environment. I-graphs show the information contents in detail. ISAC enables adding details in a systematic way, but by help of the different graphs.

We think that the data flow diagrams (SA-SD methodology) have an advantage, because it can be used for connections of the information system with the environment by sorces/sinks and for adding details by the functional decomposition and multilevel diagrams. For all this we have to use more graphical notations of the ISAC methodology.

There is an assumption in ISAC that careful and detailed decomposition of the user activities will largely procedure the information requirements. From ISAC point of view work methods are more important than description tehniques. We do not quite agree with this because we emphasize that description tehniques must be used as the basis for understanding the problem and for communication between the users and team.

TABLE 4: Table shows steps of system development of all the four discussed methodologies and the products that are produced at each step:

	working procedure: different analysis and design areas each of them is devided into more than 3 steps and substeps!)	workproducts (documentation)
ISAC:	1. CHANGE ANALYSIS:analysis of problems and needs and the current state. We define and produce alternative changes and choose the best!	A-diagrams are used for hierarchical decomposition current activities. We can use also: problem groups tables, text pages, property tables, table of objectives!
	2. ACTIVITY ANALYSIS: we continue with the decomposition of activities. We more detail define information flows and information subsystems!	A-graphs (Figure 6), property tables
	3. INFORMATION ANALYSIS: we analyse relationship between information sets and the structure of information sets.	I-graphs:hierachical graphs of information flows, textual description C-graphs
	4. DATA SYSTEM DESIGN	D-graphs D and P strctures (JSP)
	5. EQUIPMENT ADAPTATION	E- graphs
JSD:	working procedure	work products (documentation)
	1. ENTITY/ACTION STEP: We define entities and actions by help of user specifications (actions are verbs, entities are nouns).	entity action list
	2. ENTITY STRUCTURE STEP	structure hierarchical diagram (Figure 8)
	3. INITIAL MODEL STEP: An entity is defined as a proces which is with data flow or state vector conected with entity of the real world or with other process in a model. Processes are detailed described with pseudocode.	System Specification Diagram Jackson pseudokod
	4. FUNCTION STEP	System Specification Diagram
	5. SYSTEM TIMING STEP	note of temporal requirements
	6. IMPLEMENTATION STEP	System Implementation Diagram
SA-SD:	working procedure	workproducts (documentation)
	1. ANALYSIS of system actions, information flows, data bases	data flow diagrams (DFD)-(Figure 5) data dictionary, decision tables, psedocode
	2. DESIGN : with help of transform analysis and transaction analysis we produce from data flow diagram hierarchical data structure!	structure diagram psedocode
WARNIER:	working procedure	workproducts (documentation)
	subdivision the big system into several smaller, each of them have its own data procesing system.	note of subsystems
	the list of the required output and the description all of them	Warnier diagram (Figure 7)
	the organisation of all the data needed to obtain the output, the design of a logical base.	Warnier diagram
	the definition of transactions needed to update the data.	
	the definition of logical programs	Warnier diagram

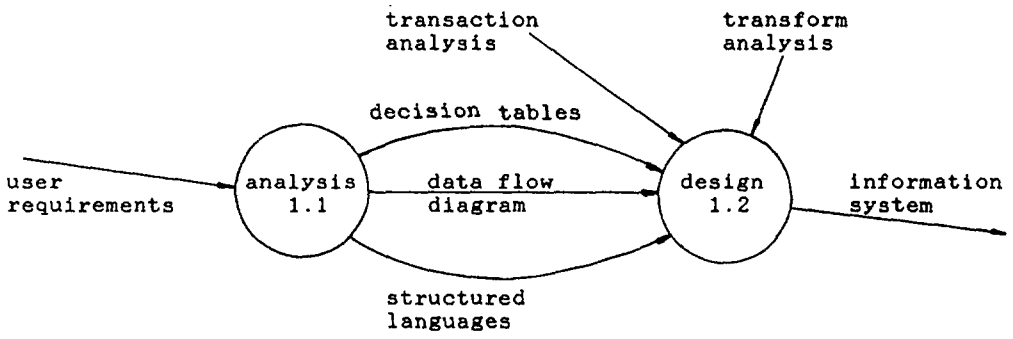


Figure 5: DFD of the SA-SD working procedure

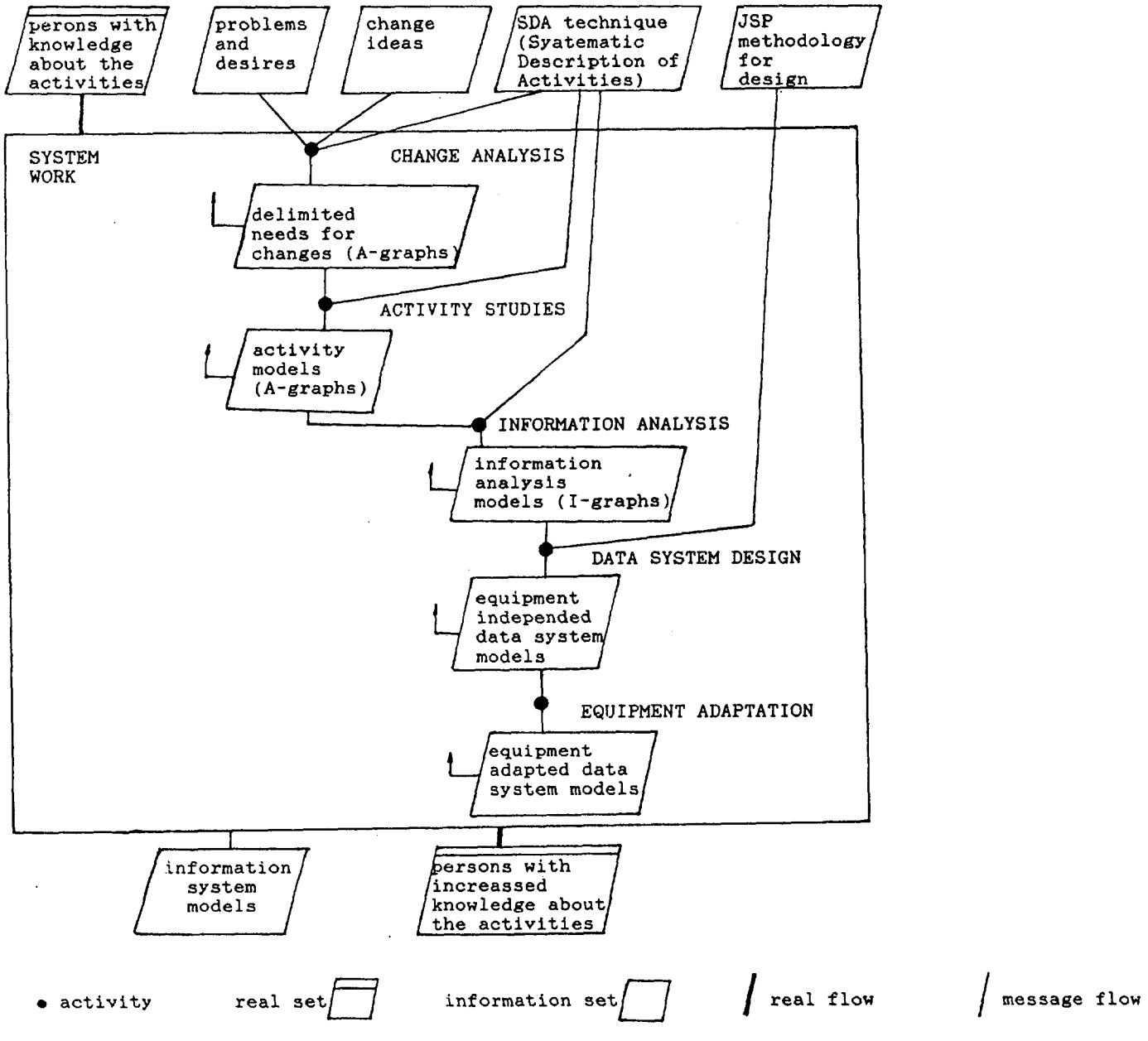


Figure 6: A- graph of the ISAC working procedure

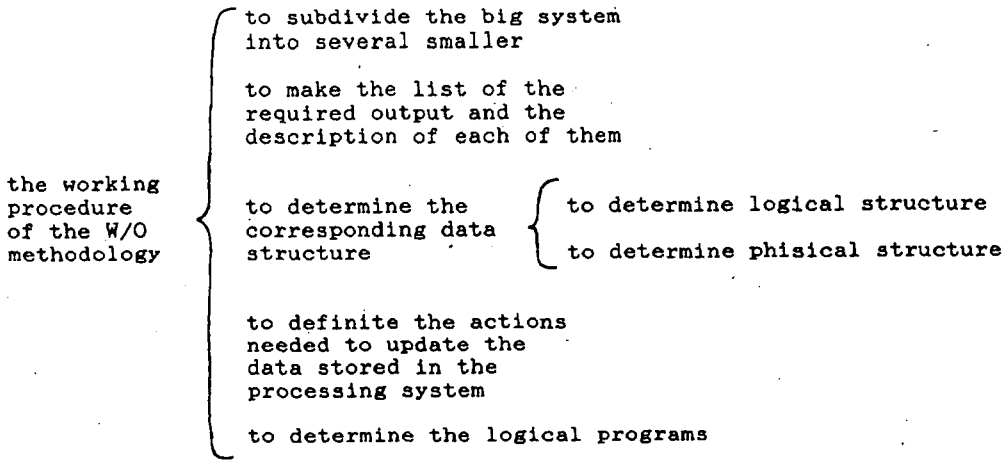


Figure 7: Warnier graph of the Warnier working procedure

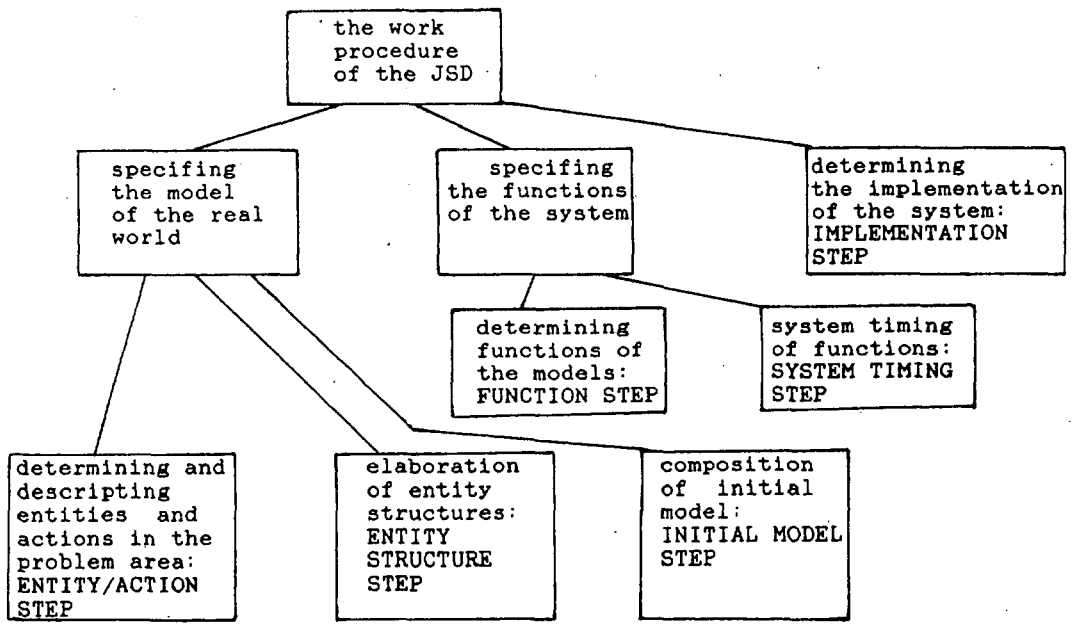


Figure 8: Structure diagram of the JSD working procedure

ISAC is relatively complex due to several levels of abstraction and several graphical notations.

It seems clear that understandability is reduced by the relative complexity of descriptions.
In all development phases of the Warnier methodology we can use only one graphical tool-Warnier diagram (Figure 7). We can describe the data and structure process by only three basic components of stuctured programming (sequence, iteration and selection).

4 BEHAVIOR IN THE REAL TIME ENVIRONMENT

Behavior of the methodology in the real-time environment is also very important because the information system represents a set of coherent different or equal actions. We think that JSD is the most suitable of all the four discussed methodologies for applications from the real world with the important temporal extension. In the JSD System Timing Step adequate measures are taken to ensure a correct scheduling of system processes. For this purpose, synhronisation processes are defined.

It is important to emhasize that JSD is intended for development not only for data processing, but for other applications also. Temporal dimension of information is not treated explicitly in ISAC, nor in the Warnier methodology.

5 LEARNABILITY

The methodology must be easy to learn because even within single organisation, there will be quite a great number of people who have to use the methodology as a resource and it must help all of them and not make a developer process more difficult.
It is clear that the methodologies must be communicable to other persons not only to developers. Learnability depends on the complexity of the methodology , which is probably related with covering the software development life cycle and perhaps on the depth of the understanding of information systems provided by it. We establish that among the four discussing methodologies only ISAC is more complex, the others are relatively simple. We think that ISAC is less easy to learn.

6 AUTOMATED TOOLS

It is clear that automated tools which offer series of understandable resources, brought near people, make possible the supervision of a project and immediate repairing existing failures. Automated tools give up to date version to all members of the team.

A great number of automated tools and environments have been explicitly developed to support nearly all studied methodologies. We do not know if such tools are commercialy available for ISAC methodology in a broad sense although in the **ISAC** group a prototype system called IA/2 was developed in the early seventies. Automated tools faciliate the work to designers and improve the productivity of both the individual developer and development team.

Tools for computer-aided software engineering provide these benefits:
- improved productivity and faster systems development (They automate design and documentation, eleminate manual drawing and redrawing and allow quick changes.)
- higher quality software (They produce universal documentacion, promote standardisation.)
- reduced maintenance (They promote easy changes and allow on-line access to design.)

7 CONCLUSION

It is clear that any information systems development methodology is better than no methodology!
We think that there is no one information systems design methodology which is best for developing all information systems.
We have represented the main findings about the methodologies. It is clear that our analysis is in many respects preliminary, because of extreme complexity of the subject.
In this section we describe importance of the methodologies from the practical point of view.

We demonstrate that each of the four methodologies is relative powerful in some situations.
We evolute the applacability of the methodologies in very crude terms by use these dimensions:

- fuzziness of the information requirements
- complexity of the data system
- complexity of the actions on the data of the system

Our view is that the relative value of ISAC are the earlier phases. It is effective and suitable for fuzzy information requirements, but it is restricted to not too much complex system because of the restrictions caused by division information and data system into relatively small and simple sub-systems without any flexible mechanism for integration.
We conclude that the Warnier methodology is less powerful in the fuzziness aspect. Then follows JSD methodology. SASD lies somewhere between JSD and ISAC which include a powerful special mechanism to manage fuzzines. (See Figure 9.a)

JSD is not quite useful for very complex data systems, but for systems with many actions, which bring entity from one stage to other, because JSD methodology provide good description technique for showing temporal sequence of actions with three basic components of structured programming (iteration, sequence, selection), but it does not provide good description of entities of entire system and relationship between them. In the implementation step we consider temporal performance of each groups of actions. It is more powerful in the description of actions then entities. (See Figure 9.b and 9.c)

We think that the Warnier methodology is suitable for developing complex data and action systems because of use only one simple and efficient Warnier/Orr diagram.

Our view is that SA-SD is very useful also for more complex data and actions systems. SA-SD methodology enables simply description of data components in the data dictionary. A structure of actions are described by a structure language or decision tables.

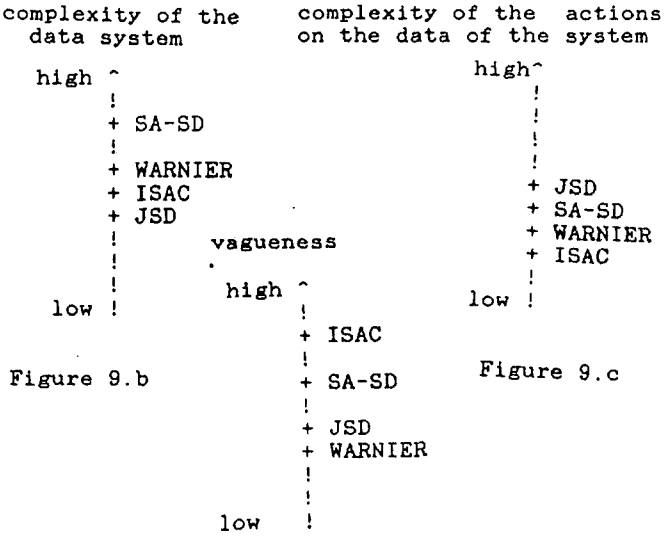


Figure 9.b

Figure 9.c

Figure 9.a

8 REFERENCES

/CAME86/ - J. R. Cameron: An overwiev of JSD, IEEE Software Engineering 1986/2.

/DEMA78/ - Tom De Marco, Structured analysis and system specification, Prentice-Hall, New Yersey, 1978.

/HIGG79/ - David A. Higgns, Program Design and Construction, Prentice-Hall, 1979.

/INFO83/ - Information Systems design methodologies: A Feature Analysis, Elsevier Science publishing company, Amsterdam, 1983.

/JACK83/ - Jackson M. A.:System Development, Prentice-Hall, 1983.

/LUND81/ - Mats Lundeberg, Goran Goldkuhl, Anders Nilsson: Information Systems Development: A System Approach, Prentice-Hall, 1981.

/PORC83/ - M. Porcella, P.Freeman, Ada Methodology Questionarie Summary, ACM Software Engineering Notes, Vol 8, No1, Januar 1983.

/THEI83/ - The Information Structures Subgroup of the Dutch Database Club: Software Engeneering: Methods and Tehniques, Wiley - Interscience Publications International 1983.

/YOUR79/ - Yourdon E., Constantine L., Structured design, Fundamentals of a Discipline of Computer Program and Design, Prentice-Hall, 1979.

UDK [681.3:002:02](479.1)

Marija Šifrar

Abstract

The first section of this paper presents a brief overview of the environment and efforts made in automating library and documentation center functions in Yugoslavia. The second part describes ongoing activities and the nation-wide project for improving services provided by libraries and documentation centers. The project is in the development stage. Some of the component programming systems are already implemented. Information services already available to the end-users are listed in the last section.

Introduction

For many years in Yugoslavia there have been made efforts to automate library and documentation center services.

No nation-wide action was undertaken in the field of library automation. There were many reasons for this. There are eight national libraries, one in each republic and province, and in addition one federal organization Yugoslav Bibliographic Institite (JBI). The library activities and library development remain the responsibility of the national libraries for each particular region. National libraries produce their respective national bibliographies, catalog cards, which they send to their subordinated libraries, to other national libraries in the country and to JBI. National libraries and JBI receive along with the catalog card also the original publication. JBI is issuing annual bibliography of Yugoslavia and is responsible for the coordination of activities with other countries. Bibliography of Yugoslavia is maintained in these nine different places.

Efforts in Automation in Libraries

The level of automation in particular libraries depended on the need for automation, the accessibility of computers, readiness to change and previous experience of staff and available finances.

Various libraries automated particular functions that represented bottlenecks in performing their everyday activities. These libraries developed programs to meet only their local needs. Software development and data processing were in most cases carried out on the leased hardware. Since all these libraries worked independantly of each other, the result was heterogeneity and incompatibility of the developed software. Nevertheless, some experience was obtained.

Libraries automated various functions, such as cataloging of bibliographic units in different formats and cataloging rules, circulation and information retrieval. Though Yugoslav libraries agreed upon the format UNIMARC and uniform cataloging rules already in 1981, in practice they rarely followed the agreement. It was very common that libraries decided to automate first the circulation and as a side-result producing online catalog in an abbreviated format for the needs of local lending and gathering of statistics. Only one library in Yugoslavia started to catalog immediately following the UNIMARC standardized format (National and University Library in Zagreb).

Programming systems developed at first, supported technical services' functions and were aimed to reduce staff effort needed to perform various repetitive library operations. Systems that support public services were also being developed later, but very limited in scope and performance.

Automation in documentation centers

With the proliferation of large information services, in other countries, especially the USA, many documentation centers were established, most of them in the north.

One of the functions of these documentation centers was the mediation of services of the large international information vendors. Obviously that could not have been done without appropriate hardware, software and communication equipment. They proceeded in two ways, one was direct online access to the host computer, and the other was copying of certain databases to the local system.

The latter method required access to a computer with appropriate software and also knowlegeable personnel who could maintain functioning of the computer, update databases, develop solutions for searching and displaying of data. They also started to build their own databases, but these were very modest in scope, the formats different and and the quality of data poor.

No documentation center had its own computer. They were renting computer equipment, which was becoming more and more expensive. Microcomputers represented, for many, a solution to the problem. Databases were getting transferred to standalone micros and thus became inaccessible online for end-users. Another difficulty with this approach, and one most detrimental, was the storage of data at various locations.

One of the activities of these documentation centers and libraries was online searching in databases of the big information vendors. Searching was done directly, using the commands of the host programming system. The communications were established through PTT network, due to the lack of better alternative, using a regular terminal equipment and modem. But as even this minimum equipment was scarce in documentation centers and libraries, so was also the interest for the information retrieval service. Usually those terminals were also used for other tasks.

Several programs have been developed for data entry and information retrieval. The most widespread system was IBIS, which was designed on the idea of similar systems developed in USA. It was implemented on different computers running various operating systems, such as DEC-10/TOPS-10, VAX/VMS, ATARI-ST/GEM. Its major disadvantage was inadequate user-interface.

Ongoing activities

It was impossible for information workers to offer efficient service and to expect major interest for their services from the users. At the beginning of 1987 a plan for automation of library and documentation center operations was proposed. Later during the year University of Maribor Computer Center (RCUM) was chosen the coordinator for library automation activities. The aim was to build an efficient system where both libraries and documentation centers could perform their tasks cooperatively.

The goal of the project is the development and implementation of a programming system supporting library technical and public services. The envisaged programminig system includes the following modules: cataloging, information retrieval, circulation and interlibray loan, and acquisitions. System support modules such as front-end interfaces, utility programs for conversions from various formats, etc. will be implemented in phases. The possibility of purchasing some of the modules is not excluded.

The project includes the installation of communication and computer infrastructure with the following components:

- host computer (VAX 8800, location RCUM),
- subsystems for larger users (uVAX or a more powerful VAX computer),
- intelligent terminals (ATARI or IBM PC),
- dumb terminals (VT100 compatible),
- leased PTT lines, and LANs.

Communication between the systems will be realized via packet switched network JUPAK (X.25 protocol).

In most cases computer, terminal and communications equipment will be supplied to libraries and documentation centers by the RCUM.

In 1987, three cataloging departments at the University of Maribor Library (UKM), National and University Library in Ljubljana, Yugoslav Bibliographic Institute in Belgrade, obtained the basic software, hardware and communications support from RCUM, and started to maintain the union catalog in a shared cataloging mode. They input bibliographic records for monographs and periodical article citations for the national bibliography in the UNIMARC format.

Several databases, built at other sites in different formats, were converted to UNIMARC format and installed at the host computer at RCUM.

Besides shared cataloging module, UKM and NUK started to use a new version of circulation system. The searching module for the bibliographic databases is presently implemented with basic capabilities and the major effort in the software development area will be devoted to this module.

Information Services

By the end of 1987 RCUM started to provide information services to the end-users by enabling online access and searching for the data in the databases built in Yugoslavia and installed on the host system: online union catalog, international ISDS (Paris) database for periodical publications, specialized databases for medicine, textile industry, mechanical engineering, economics. Local software is used for information retrieval.

By December 1987 the communication with and access to the information services and library networks outside the country was provided via the packet switched network JUPAK (x.25 protocol): DIALOG, WLN, OCLC - USA; DATASTAR, DATAMAIL - Switzerland; FRASCATI - Italy; ECHO - Luxemburg; CAS/STN, DIMDI, Informacion center PTT MANNHEIM - FR Germany, MIC - Sweden.

Conclusion

Thus I have hopefully presented our first steps on the way to meeeting the major challenge of the next century, effective resource sharing and data transfer between information providers and information users.

UDK 681.3.06 SYSTRAL:519.688

Hrvoje Nežić
Elektrotehnički institut »Rade Končar«, Zagreb

Članak daje prikaz hibridnog jezika SYSTRAL, koji predstavlja nadgradnju konvencionalnog zbirnog jezika. Njegova sintaksa, uz male prilagodbe, može se primijeniti na zbirni jezik bilo kojeg procesora. Predstavljena je i formalna definicija verzije jezika za procesor MC68000 u vidu LL(1) gramatike, uz više primjera za ilustraciju. Razvoj jezika nije potpuno dovršen.

TOWARD A SYMBOLIC STRUCTURED ASSEMBLY LANGUAGE. This article describes SYSTRAL, a hybrid language, which is a superstructure of the conventional assembly language. The paper deals with the MC68000 version of the language. However, its syntax can be applied to the assembly language of any processor, with only a few adjustments. The formal definition of the language as an LL(1) grammar is also presented, as well as a number of illustrative examples. The development of the language has not still been finished.

1. UVOD

Pisanje programa u zbirnom jeziku još uvijek nije mrtva umjetnost, zbog značajnih ušteda u brzini i memorijskom prostoru /MORTON86/ , pa zato ima smisla nastojanje da se taj rad učini produktivnijim. U članku /NEŽIĆ85/ opisao sam svoje prve ideje o strukturiranom zbirnom jeziku, proizašle iz težnje da si olakšam rad na relativno velikom programskom projektu u zbirnom jeziku sa kojim sam se suočio. U to vrijeme nisam znao za druge slične radove: /WALKER81A/, /WALKER81B/, /KRIEGER80/, /WIRTH68/, /KAWAI80/, /ANDERSON81/. Moj pristup rješavanju istog problema bio je donekle drugačiji od pristupa u tim radovima i omogućio je razvoj i obogaćivanje osnovnih ideja, što je rezultiralo kompleksnijim mehanizmom strukturiranja koji se može nazvati i jezikom.

SYSTRAL (SYmbolic STRuctured Assembly Language) je hibridni jezik baziran na običnom zbirnom jeziku. Programi se u njemu grade miješanjem strukturnih instrukcija i običnih instrukcija zbirnog jezika i zbog toga ga možemo promatrati kao jezik koji pored ključnih riječi, identifikatora, itd., sadrži još jedan leksički element – blok običnih zbirnih instrukcija. Na viši nivo podignuta je samo jedna komponenta zbirnog jezika, a to je upravljanje tokom programa. Promatrajući odnos SYSTRAL-a prema zbirnom jeziku može se na primjer povući određena analogija sa odnosom također hibridnog jezika Objective C prema njegovom baznom jeziku C. Međutim, C, pa tako i Objective C, je jedinstven jezik, dok su zbirni jezici raznih procesora različiti, i razlike prevelike a da bi SYSTRAL mogao biti sasvim jedinstven jezik. Ipak, svi osnovni koncepti jednaki su za sve procesore, a i razlike u sintaksi su vrlo malene.

Općenito, postoje tri načina implementacije strukturiranog zbirnog jezika: implementacija pomoću makroinstrukcija, implementacija pomoću pretprocesora, i cjelokupna implementacija strukturiranog zbirnog jezika. SYSTRAL se može implementirati drugim i trećim načinom, dok se pomoću makroinstrukcija može implementirati samo njegov podskup. U vrijeme pisanja ovog članka implementiran je kompilator u obliku pretprocesora za komercijalno dobavljiv zbirnik procesora MC68000.

2. KRATAK PREGLED SVOJSTAVA JEZIKA

Prije nego pređemo na detaljniji opis jezika dat ćemo sažetak njegovih svojstava.

SYSTRAL je jezik koji sadrži naredbe za strukturirano upravljanje tokom programa. Odabir struktura je takav da uz preglednost, čitljivost i lakoću programiranja omogućuje i efikasnost. Uz uključivanje optimizacionih algoritama u prevodilačke mehanizme dobiveni kod gotovo u svim slučajevima može biti jednako efikasan kao onaj dobiven direktno u zbirnom jeziku, pri čemu je preglednost programa i lakoća kojom se mogu provoditi izmjene u programu neusporedivo veća.

Definirana su dva nivoa jezika: SYSTRAL-1 (može se implementirati jednoprolaznim pretprocesorom) i SYSTRAL-2 (za implementaciju je neophodan dvoprolazni pretprocesor). Prvi nivo je podskup drugog.

Programiranje se vrši miješanjem strukturiranih i običnih zbirnih programskih linija.

Logički uvjeti baziraju se na ispitivanju status bitova procesora, no takvi osnovni uvjeti mogu se povezivati u složene logičke izraze korištenjem AND_ i OR_ operatora, a na drugom nivou jezika u logičkim izrazima mogu se pojavljivati i zagrade. Nazivi stanja status bitova sastoje se od cijelih riječi ili

dužih skraćenica, a za razna značenja istog stanja postoje i razni nazivi.

Detalji u oblicima ciljnih instrukcija koje će generirati kompilator mogu se kontrolirati u izvornom tekstu korištenjem modifikatora za modifikaciju njihovih unaprijed definiranih oblika.

Registrima procesora mogu se pridjeljivati simbolički nazivi, tj. simbolički opisi njihovih trenutnih značenja.

SYSTRAL je u velikoj mjeri prenosiv na razne procesore, uz korištenje gotovo iste sintakse. Razlike za razne procesore očituju se uglavnom u različitim modifikatorima, različitim argumentima u FOR_, SWITCH_ i CASE_ naredbama i različitim nazivima stanja status bitova.

3. DETALJNI OPIS JEZIKA

Jezik, tj. njegovu verziju za procesor MC68000, ovdje ćemo prikazati služeći se opisnim, dakle neformalnim načinom, te formalnim, u vidu LL(1) gramatike, pri čemu se oba prikaza uzajamno dopunjuju.
Opišimo najprije notaciju kojom ćemo se koristiti u prikazu gramatike.

3.1 Notacija za prikaz sintakse

Svi završni simboli (ključne riječi, interpunkcije itd.) prikazuju se kao identifikatori sastavljeni od velikih slova i znaka "_" (npr. IF_, ENDIF_, ZAREZ, DATA_REG).

Svi nezavršni simboli prikazuju se kao identifikatori sastavljeni od malih slova i znaka "_" (npr. program, lista_naredbi, faktor).

Format prikaza produkcija gramatike je slobodan, osim u jednom elementu: nezavršni simboli na lijevoj strani produkcija uvijek započinju na samom lijevom rubu prikaza, dok svi simboli sa desne strane produkcija započinju sa određenim pomakom od lijevog ruba prikaza:

```
lijeva_strana   --> prvi_simbol drugi_simbol
                    treći_simbol
```

Sve produkcije sa jednakom lijevom stranom nalaze se na jednom mjestu i zapisuju skraćeno - simbol sa lijevih strana tih produkcija pojavljuje se samo jednom, prije zapisa desnih strana:

```
nez_simbol  --> desna_strana_prve_produkcije
            --> desna_strana_druge_produkcije
                    ....
            --> desna_strana_zadnje_produkcije
```

Ako je desna strana produkcije prazna, to se označava navođenjem simbola EPS (skraćenica od "epsilon"), npr.:

```
nez_simbol  --> desna_strana_prve_produkcije
            --> EPS
```

3.2 Organizacija programskih linija

Programe u SYSTRAL-u gradimo miješajući obične instrukcije zbirnog jezika (uključujući i direktive) sa instrukcijama strukturiranja. Miješanje se vrši na nivou programskih linija. Dakle, uvijek razlikujemo običnu zbirnu programsku liniju i strukturiranu programsku liniju. Svaka strukturna programska linija ima oblik sličan kao zbirna linija:

(labela) ključna_riječ (parametri) (komentar)

(elementi navedeni unutar zagrada mogu izostati).

Iako se na početku strukturne linije može nalaziti labela, ona se u općem slučaju zanemaruje, pri čemu postoji nekoliko izuzetaka.

3.3 Leksički elementi

Između pojedinih leksičkih elemenata unutar strukturnih programskih linija mogu se proizvoljno pojavljivati praznine i tabulatori. Ukoliko nisu neophodni za razdvajanje susjednih elemenata oni se zanemaruju.
Prilikom neformalnog opisa leksičkih elemenata jezika navodit ćemo i njihove simbole koji se koriste u formalnom prikazu sintakse.

3.3.1 Ključne riječi

Sve ključne riječi su rezervirane i sadrže na kraju znak "_" da bi se izbjegla mogućnost kolizije prvenstveno sa direktivama zbirnog jezika. Tako se npr. ključne riječi IF_, ELSEIF_, ELSE_ i ENDIF_ zahvaljujući znaku "_" razlikuju od uobičajenih direktiva IF, ELSEIF, ELSE i ENDIF. SYSTRAL sadrži slijedeće ključne riječi:

IS_	OR_	AND_	NOT_
IF_	ELSEIF_	ELSE_	ENDIF_
LOOP_	WHILE_	REPEAT_	
FOR_	NEXT_		
EVENT_	UNTIL_		
MONITOR_	WHEN_	RESUME_	
SWITCH_	CASE_	ENDSWITCH_	

Simboli koji se koriste za prikaz ključnih riječi u opisu sintakse jednaki su samim ključnim riječima.

3.3.2 Nazivi stanja status bitova

Sa ciljem povećanja čitljivosti programa zamišljeno je ovakvo tretiranje naziva stanja status bitova:

- za određeno stanje status bita može postojati više od jednog naziva, jer pojedina stanja mogu imati više značenja ovisno o kontekstu u kojem se ispituju

- svaki naziv umjesto samo jednog ili dva slova sadrži cijelu riječ ili barem dužu skraćenicu.

U tablicama 1 i 2 dani su nazivi stanja status bitova za procesore MC68000 i Z80. Ako izuzmemo nazive EVEN i ODD, preostali nazivi stanja status bitova procesora Z80 su podskup naziva procesora MC68000.
Slovo "U" u nazivima LESS_U, GREATER_U itd. označava da se radi o brojevima bez predznaka (unsigned).

U formalnom opisu sintakse za leksičku kategoriju naziva stanja status bitova upotrebljava se simbol UVJET.

SYSTRAL	zbirni jezik
EQUAL ZERO FALSE	EQ
NOTEQUAL NOTZERO TRUE	NE
LESS	LT
GREATER	GT
LESSEQUAL	LE
GRTEQUAL	GE
LESS_U CARRY	CS
GREATER_U	HI
LESSEQUAL_U	LS
GRTEQUAL_U NOTCARRY	CC
POSITIVE	PL
NEGATIVE	MI
OVERFLOW	VS
NOTOVERFLOW	VC

Tablica 1. Nazivi stanja status bitova procesora MC68000

SYSTRAL	zbirni jezik
EQUAL ZERO FALSE	Z
NOTEQUAL NOTZERO TRUE	NZ
LESS_U CARRY	C
GRTEQUAL_U NOTCARRY	NC
POSITIVE	P
NEGATIVE	N
EVEN OVERFLOW	PE
ODD NOTOVERFLOW	PO

Tablica 2. Nazivi stanja status bitova procesora Z80

3.3.3 Modifikatori

Modifikatori služe prvenstveno zato da bi se u izvornom kodu mogli kontrolirati detalji u oblicima ciljnih instrukcija koje će generirati kompilator, tj. za eksplicitno modificiranje implicitnih oblika tih instrukcija.

Kao primjer možemo navesti modifikator dugog skoka, budući da su instrukcije skokova najčešće ciljne instrukcije. Skokovi mogu biti kratki i dugi, a ako nije eksplicitno drugačije specificirano mehanizmi translatiranja implicitno generiraju efikasnije kratke skokove. Ukoliko je na nekom mjestu u programu umjesto kratkog potreban dugi skok, željeni efekt postiže se ubacivanjem modifikatora dugog skoka na to mjesto.

Zbog uske povezanosti sa instrukcijama ciljnog zbirnog jezika, u broju, značenjima i sintaksnim pravilima za upotrebu modifikatora koncentrirana je većina razlika u verzijama SYSTRAL-a za razne procesore.

Svi modifikatori sastoje se od jednog ili više znakova unutar vitičastih zagrada, pri čemu nije dozvoljena upotreba praznina. Slijedeća tablica daje pregled svih modifikatora u verziji za procesor MC68000, njihovih simbola u formalnom opisu sintakse, te riječi čija su početna slova sastavni dijelovi modifikatora:

Modifikator	Simbol	Izvor
{L}	MODIF_L	Long
{B}	MODIF_B	Byte
{A}	MODIF_A	Address
{Q}	MODIF_Q	Quick
{-1}	MODIF_M	

Značenja pojedinih modifikatora bit će opisana kasnije.

3.3.4 Identifikatori i konstante

Budući da je SYSTRAL hibridni jezik i najlakše sa je implementirati u obliku pretprocesora za neki postojeći zbirnik, a razni zbirnici sadrže i različita pravila za tvorbu identifikatora, numeričkih i znakovnih konstanti, on ne određuje jedinstvena pravila za tvorbu tih leksičkih elemenata. U slučaju implementacije u obliku pretprocesora preuzimaju se pravila konkretnog baznog zbirnog jezika.

U formalnom opisu sintakse za ove leksičke elemente upotrebljavamo simbole IDENT, NUM_KONST i ZN_KONST.

3.3.5 Registri

Jedna od specifičnosti SYSTRAL-a je mogućnost pridjeljivanja simboličkih naziva registrima procesora. Riječ "simbolički" u nazivu jezika odnosi se upravo na ovu mogućnost.

Zbirni jezici omogućavaju korištenje simbola samo za označavanje memorijskih lokacija i konstanti, dok najčešće korišteni objekti - registri - imaju standardne nazive koji međutim ne opisuju trenutna značenja sadržaja registara, i ne postoji način izražavanja takvih značenja u izvornom tekstu, osim u komentarima.

U SYSTRAL-u se svakom registru za kojeg je predviđena ta mogućnost može na bilo kojem mjestu u programu pridijeliti simbolički opis njegovog trenutnog značenja, jednostavno tako da se standardnom imenu registra doda znak "_" kojeg slijedi opis (npr. D1_LENGTH, D5.W_OFFSET, A0_TABLE). Pravila tvorbe ovakvih simboličkih opisa jednaka su pravilima tvorbe identifikatora. Simbolička imena registara mogu se pojavljivati i u strukturnim i u zbirnim programskim linijama.

Izvedba za procesor MC68000 u strukturnim programskim linijama dozvoljava korištenje podatkovnih registara, adresnih registara i programskog brojila. Ove leksičke kategorije u formalnom opisu sintakse imaju simbole DATA_REG, ADR_REG i PC_REG. Simbolička imena mogu se pridjeljivati samo podatkovnim i adresnim registrima.

3.3.6. Supstitucije makro argumenata

Ukoliko se neka struktura jezika koja zahtijeva određene argumente u obliku registara, konstanti itd. (FOR_ ili SWITCH_ struktura) nalazi unutar makro definicije, može se pojaviti potreba da se kao argument u strukturi umjesto fiksnog registra ili konstante pojavi formalni argument makro definicije.

Kao i za identifikatore i konstante, SYSTRAL ne određuje ni pravila tvorbe makro argumenata, već se kod implementacije pretprocesorom preuzimaju pravila baznog jezika. Formalni argumenti makro-a u raznim zbirnim jezicima pojavljuju se u dva oblika: pozicionom (specificira se redni broj argumenta, npr. "1") ili simboličkom. U prvom slučaju potreban je posebni leksički element koji ovdje nazivamo supstitucijom makro argumenta i označavamo simbolom SUPST, dok u drugom slučaju formalni argumenti ulaze u kategoriju identifikatora. Razlike između ova dva slučaja dovode i do neznatnih razlika u sintaksi jezika. U ovom radu pretpostavljamo prvi slučaj i predstavljamo odgovarajuću gramatiku.

3.3.7 Specijalni znakovi

Slijedeća tabela daje popis svih specijalnih znakova u izvedbi jezika za procesor MC68000 i njihovih simbola za korištenje u formalnom opisu sintakse.

Znak	Simbol
,	ZAREZ
(	L_ZAG
)	D_ZAG
#	OZNAKA_BROJA
.	TOČKA
-	MINUS
+	PLUS

3.3.8 Blokovi zbirnih instrukcija

Zbog principa miješanog kodiranja, blok od jedne ili više zbirnih programskih linija uključujući i potpuno prazne čini jedan leksički element kojeg označavamo simbolom ASM_BLOK.

3.3.9 Završeci strukturnih linija

Iz istog razloga neophodno je i postojanje ovog leksičkog elementa. On se može sastojati samo od znaka za završetak linije (npr. CR), ali i od komentara kojeg slijedi CR. Označavamo sa simbolom NEWLINE.

3.4 Sintaksa i semantika jezika

3.4.1 Prikaz programa

U poglavlju 3.4 dat ćemo uz ostalo i veći broj primjera strukturiranih programa. Ispis nekih primjera dobiven je primjenom programa koji automatski uokviruje sve strukture, koristeći znakove "!","_" i "-", npr.:

```
!----------------------!
! IF_                  !
!       !--------------!!
!       ! LOOP_        !!
!       !      ...     !!
!       ! REPEAT_      !!
!       !--------------!!
! ENDIF_               !
!----------------------!
```

Time je povećana preglednost prikaza strukturiranih programa.

3.4.2 Struktura programa

Prije nego pređemo na opis pojedinih dijelova jezika, utvrdimo kako može izgledati legalan program. Svaki program ima oblik liste, tj. niza naredbi, pri čemu svaka naredba može biti složena (jedna od struktura jezika) ili jednostavna (blok zbirnih instrukcija ili strukturna instrukcija EVENT_). Ovdje su prikazani sintaksni elementi (program), (lista_naredbi), i (naredba). Element (lista_naredbi) često se koristi i unutar pojedinih struktura, kao što ćemo vidjeti u daljnjem tekstu.

```
program          -->   lista_naredbi

lista_naredbi    -->   naredba lista_naredbi
                 -->   EPS

naredba          -->   ASM_BLOK
                 -->   event_naredba
                 -->   if_struktura
                 -->   loop_struktura
                 -->   for_struktura
                 -->   monitor_struktura
                 -->   switch_struktura
```

3.4.3 Logički izrazi

U SYSTRAL-u se mogu tvoriti složeni logički izrazi upotrebom AND_ i OR_ operatora. Međutim, osnovni logički uvjeti su, kao i u običnom zbirnom jeziku, zasnovani na status bitovima. Opišimo najprije kako se specificiraju osnovni logički uvjeti.

3.4.3.1 Instrukcija IS_

U zbirnim jezicima ispitivanje određenog uvjeta provodi se u dva koraka:

1) obnavljanje status bita
2) ispitivanje stanja status bita
   i provođenje grananja ovisno o stanju.

Zbog toga SYSTRAL omogućava podržavanje obje ove faze. Razdvajanje tih dviju faza u svim upravljačkim strukturama jezika omogućuje instrukcija IS_ .
Na primjer, osnovni oblik IF_ strukture izgleda ovako:

```
IF_
        (lista_naredbi_1)
IS_ (uvjet)
        (lista_naredbi_2)
ENDIF_
```

Element (lista_naredbi_1) i instrukcija IS_ (uvjet) zajedno čine najjednostavniji oblik logičkog izraza - (lista_naredbi_1) obnavlja status bit, a instrukcija IS_ (uvjet) sa ispituje i ovisno o njesovom stanju provodi srananje.
Prethodnu konstrukciju treba shvatiti i čitati na slijedeći način:

```
Ako
      obnavljanje status bita
rezultira određenim stanjem status bita
      tada izvrši listu naredbi 2.
```

Pritom je (lista_naredbi_1) primjer sintaksnog elementa (lista_naredbi) opisanog u prethodnom poglavlju, a (uvjet) se sastoji od leksičkog elementa UVJET kojemu može prethoditi NOT_ operator (leksički element UVJET predstavlja nazive stanja status bitova). Element (lista_naredbi) može biti sastavljen i od složenih struktura, no u funkciji obnavljanja status bitova on će se najčešće reducirati jednostavno u blok zbirnih instrukcija ili čak u praznu listu naredbi. No bez obzira na sastav (liste_naredbi_1) prevodilački program zamijenit će ključnu riječ ENDIF_ labelom, a instrukciju IS_ (uvjet) uvjetnim skokom na tu labelu.

Za ilustraciju kodirajmo jednostavni zadatak - ako su resistri D0 i D1 jednaki tada u resistar D2 treba upisati broj 1:

```
IF_
     CMP.L  D0,D1
IS_ EQUAL
     MOVE.L #1,D2
ENDIF_
```

Prevođenjem ovog odsječka dobiva se slijedeći ekvivalentni zbirni program (zbog veće preslednosti uključene su i izvorne instrukcije u obliku komentara):

```
                        ; IF_
        CMP.L    D0,D1
        BNE.S    LAB1       ; IS_ EQUAL
        MOVE.L   #1,D2
LAB1    ....                ; ENDIF_
```

Pošto se ključna riječ IF_ ne translatira u nikakav kod, odsječak za obnavljanje status bita može joj i prethoditi umjesto da se nalazi između nje i ključne riječi IS_. Tako je slijedeći program potpuno ekvivalentan prethodnom:

```
CMP.L  D0,D1
IF_
IS_ EQUAL
     MOVE.L #1,D2
ENDIF_
```

Instrukcija IS_ predstavlja u SYSTRAL-u jedini način za eksplicitno izražavanje primitivnih uvjeta baziranih na status bitovima, a primijenjuje se u povezanosti sa svim instrukcijama koje zahtijevaju izražavanje takvih uvjeta (u gornjim primjerima to je bila instrukcija IF_), najčešće po ovom obrascu:

```
(ključna_riječ)
        (lista_naredbi)
IS_  (uvjet)
```

gdje (ključna_riječ) može biti IF_, ELSEIF_, WHILE_ ili EVENT_.

Slijedeći primjer ilustrira korištenje instrukcije IS_ u povezanosti sa ključnom riječi ELSEIF_ i ujedno prikazuje slučaj u kojem je neophodno da lista naredbi za obnavljanje status bita bude prazna. Zadatak je slijedeći: u resistar D1 treba upisati broj 0, 1, ili 2, ovisno o tome da li je riječ u resistru D0 jednaka nuli, nesativna ili pozitivna.

```
IF_
      TST  D0
IS_  ZERO
      MOVE #0,D1
ELSEIF_
IS_  NEGATIVE
      MOVE #1,D1
ELSE_
      MOVE #2,D1
ENDIF_
```

Odsječak za obnavljanje status bita između ključnih riječi ELSEIF_ i IS_ u ovom slučaju je prazan, no ponekad se može javiti potreba da on sadrži neku od struktura jezika, kao u slijedećem zadatku - u resistar D1 treba upisati broj 0, 1, ili 2 ovisno o tome da li je apsolutna vrijednost riječi u resistru D0 jednaka 0, veća od 100, odnosno manja ili jednaka 100:

```
IF_
      TST  D0
IS_  ZERO
      MOVE  #0, D1
ELSEIF_
      IF_
      IS_  NEGATIVE
           NEG  D0
      ENDIF_
      CMP  #100, D0
IS_  GREATER
      MOVE  #1, D1
ELSE_
      MOVE  #2, D1
ENDIF_
```

Osim u IF_ strukturi instrukcija IS_ upotrebljava se i u drusim strukturama jezika. U slijedećem primjeru ona se pojavljuje u strukturi petlje sa izlazom na dnu. Program služi za računanje broja nesativnih bytova u bloku određenom resistrima A0 (početna adresa) i A1 (završna adresa).

```
CLR.W   D0_COUNT
LOOP_
     IF_
          TST.B  (A0_TREN_ADR)+
     IS_  NEGATIVE
          ADDQ.W  #1, D0_COUNT
     ENDIF_
REPEAT_  WHILE_
     CMPA.L  A1_ZAV_ADR, A0_TREN_ADR
IS_  LESSEQUAL_U
```

Nazivu stanja status bita u IS_ instrukciji može prethoditi operator NOT_ koji specificira suprotno stanje istog status bita. Npr. instrukcija IS_ NOT_ TRUE je ekvivalentna instrukciji IS_ FALSE.

3.4.3.2 AND_ i OR_ operatori

Instrukcija IS_ u sprezi sa odsječkom za obnavljanje status bita koji joj prethodi predstavlja najjednostavniji oblik logičkog izraza. Složeniji izrazi tvore se povezivanjem ovakvih osnovnih pod-izraza pomoću AND_ i OR_ operatora, a implementiraju se načinom koji se ponekad naziva kaskadama skokova /WAITE84/. Uz pretpostavku da izraz sadrži dva operanda (pod-izraza) povezanih AND_ ili OR_ operatorom, takav način implementacije znači da se evaluacija izraza prekida nakon evaluacije prvog operanda ako ona daje dovoljnu informaciju da se odredi rezultat cijelog izraza. Pravila su slijedeća:

- ako izraz sadrži dva uvjeta povezana AND_ operatorom i prvi uvjet nije ispunjen, tada nije ispunjen ni uvjet čitavog izraza

- ako izraz sadrži dva uvjeta povezana OR_ operatorom i prvi uvjet je ispunjen, tada je ispunjen i uvjet čitavog izraza.

U višim jezicima koji definiraju evaluaciju logičkih izraza na ovakav način, logički izrazi se evaluiraju slijeva nadesno, dok se u SYSTRAL-u može govoriti o evaluaciji odozgo prema dolje, s obzirom da se logički izrazi razvijaju vertikalno, a ne horizontalno.

Definicija logičkih izraza razlikuje se za prvi i drugi nivo jezika. Prvi nivo predviđen je za jednostavniju implementaciju jednoprolaznim pretprocesorom pa su stoga nužna određena ograničenja. Sintaksno su logički izrazi na prvom nivou jezika podskup onih na drugom, no i za sintaksno iste izraze, na raznim nivoima semantika se može razlikovati.

Prije opisa logičkih izraza za pojedine nivoe definirajmo sintaksne elemente (not_operator) i (jmpmodif) (modifikator dugog skoka) zajedničke za oba nivoa:

```
not_operator      -->   NOT_
                  -->   EPS

jmpmodif          -->   MODIF_L
                  -->   EPS
```

3.4.3.2.1 AND_ i OR_ operatori - nivo 1

Na prvom nivou jezika AND_ i OR_ operatori imaju jednak prioritet, upotreba zagrada u logičkim izrazima nije dozvoljena, osnovni logički uvjeti podliježu pravilu desne asocijativnosti (odnosno asocijativnosti odozdo prema gore), a NOT_ operator može se koristiti samo za negaciju naziva stanja status bitova.
Nepostojanje eksplicitnog grupiranja pomoću zagrada, jednaki prioriteti operatora i pravilo desne asocijativnosti daju efekt kao da se nakon svakog AND_ ili OR_ operatora nalazi otvorena zagrada, a sve zagrade se zatvaraju na kraju izraza.
Logički izraz može se sastojati od proizvoljno mnogo IS_ instrukcija, pri čemu svakoj prethodi lista naredbi za obnavljanje status bita, a svaka IS_ instrukcija osim posljednje u izrazu sadrži AND_ ili OR_ operator. Sintaksa je jednostavna:

```
logički_izraz     --)   NEWLINE lista_naredbi
                        IS_ not_operator UVJET
                        jmpmodif ostatak_izraza

ostatak_izraza    --)   AND_  logički_izraz
                  --)   OR_   logički_izraz
                  --)   EPS
```

Za primjer kodirajmo slijedeći zadatak - treba pozivati potprogram POTP koji vraća rezultat u registru D0, sve dok je rezultat u granicama između 50 i 100:

```
LOOP_
        BSR  POTP
REPEAT_ WHILE_
        CMP  #50, D0
IS_ NOT_ LESS          AND_
        CMP  #100, D0
IS_ NOT_ GREATER
```

Logički izraz u slijedećem primjeru je nešto složeniji - ako je ispunjen uvjet kojeg bi u nekom višem jeziku specificirali kao

```
u1 and not u2 and ( u3 or u4 )
```

tada treba komplementirati u5 (u1, u2, .. , u5) su memorijske lokacije sa značenjem logičkih varijabli):

```
IF_
        TST.B  U1
IS_ TRUE           AND_
        TST.B  U2
IS_ NOT_ TRUE      AND_
        TST.B  U3
IS_ TRUE           OR_
        TST.B  U4
IS_ TRUE
        NOT.B  U5
ENDIF_
```

Svakom logičkom izrazu pridružene su dvije labele: labela koja odgovara ispunjenom uvjetu izraza i labela koja odgovara neispunjenom uvjetu izraza. Prevodilački mehanizmi svaku IS_ instrukciju zamijenjuju instrukcijom uvjetnog skoka, pri čemu se razlikuju tri slučaja:

- ako IS_ instrukcija sadrži AND_ operator generira se instrukcija uvjetnog skoka na labelu neispunjenog uvjeta (uvjet skoka suprotan je uvjetu u IS_ instrukciji)

- ako IS_ instrukcija sadrži OR_ operator generira se instrukcija uvjetnog skoka na labelu ispunjenog uvjeta (uvjet skoka jednak je uvjetu u IS_ instrukciji)

- ako IS_ instrukcija ne sadrži AND_ ili OR_ operator moguća su oba gornja oblika instrukcije uvjetnog skoka, što ovisi o kontekstu u kojem se nalazi logički izraz.

Kompilator implicitno generira kratke skokove, no oni se mogu eksplicitno modificirati u duge primjenom modifikatora dugog skoka (L) .
U logičkim izrazima on se uvijek ubacuje iza naziva status bita u IS_ instrukciji:

```
IF_
    ...
IS_  EQUAL  (L)  OR_
    ...
IS_  ZERO  (L)
    ...
ENDIF_
```

3.4.3.2.2 AND_ i OR_ operatori - nivo 2

Na drugom nivou jezika AND_ operator ima veći prioritet nego OR_, u logičkim izrazima je dozvoljena upotreba zagrada uz proizvoljnu dubinu gniježđenja, a NOT_ operator može se koristiti i za negaciju čitavih složenih pod-izraza.

Kao i na prvom nivou, izraz može sadržavati više IS_ instrukcija od kojih samo zadnja nema AND_ ili OR_ operator, a svakoj prethodi odsječak za obnavljanje status bita. Naravno, sintaksa je nešto složenija:

```
logički_izraz   -->  član ostali_članovi

član            -->  faktor ostali_faktori

ostali_članovi  -->  OR_ član ostali_članovi
                -->  EPS

faktor          -->  NEWLINE lista_naredbi
                     IS_ not_operator UVJET jmpmodif

                -->  not_operator L_ZAG
                     logički_izraz D_ZAG

ostali_faktori  -->  AND_ faktor
                     ostali_faktori

                -->  EPS
```

Ilustrirajmo sintaksu na nekoliko primjera. U prvom primjeru kodirat ćemo izraz

```
( u1 or u2 ) and ( u3 or u4 ) :

    IF_ (
          TST.B  U1
    IS_ TRUE         OR_
          TST.B  U2
    IS_ TRUE     )   AND_  (
          TST.B  U3
    IS_ TRUE         OR_
          TST.B  U4
    IS_ TRUE         )
          NOT.B  U5
    ENDIF_
```

Ovaj odsječak translatira se u slijedeći program u zbirnom jeziku:

```
                        ; IF_ (
        TST.B   U1
        BNE.S   LAB1    ; IS_ TRUE OR_
        TST.B   U2
        BEQ.S   LAB3    ; IS_ TRUE ) AND_ (
LAB1    TST.B   U3
        BNE.S   LAB2    ; IS_ TRUE OR_
        TST.B   U4
        BEQ.S   LAB3    ; IS_ TRUE )
LAB2    NOT.B   U5
LAB3      ...           ; ENDIF_
```

Kao što se vidi iz primjera, pored labela ispunjenog i neispunjenog uvjeta ciljevi pojedinih uvjetnih skokova mogu biti i labele generirane unutar izraza (u ovom slučaju LAB1). To je i razlog zašto se ovaj način implementacije logičkih izraza naziva kaskadama skokova.

U slijedećem primjeru kodirat ćemo izraz

```
(u1 or u2 and not (u3 or u4)) and u5 :

    IF_ (
          TST.B  U1
    IS_ TRUE         OR_
          TST.B  U2
    IS_ TRUE         AND_  NOT_ (
          TST.B  U3
    IS_ TRUE         OR_
          TST.B  U4
    IS_ TRUE         ) )  AND_
          TST.B  U5
    IS_ TRUE
          NOT.B  U6
    ENDIF_
```

Gornji primjer ilustrira upotrebu NOT_ operatora za negaciju složenog pod-izraza.

Implementacija se temelji na de Morganovim teoremima - OR_ operatori se tretiraju kao AND_ i obratno, uz negaciju operanada.

3.4.4 IF_ struktura

U dosadašnjem tekstu naveli smo više primjera sa ovom strukturom. Osim što uključuje specifične logičke izraze ona je slična odgovarajućim strukturama u višim jezicima (npr. u jeziku ADA) i zato nema potrebe za njenim detaljnim opisivanjem. Najpreciznije je opisuje formalni prikaz sintakse:

```
if_struktura    -->  IF_ logički_izraz NEWLINE
                     lista_naredbi elseif_lista
                     else_blok ENDIF_ NEWLINE

elseif_lista    -->  ELSEIF_ jmpmodif
                     logički_izraz NEWLINE
                     lista_naredbi elseif_lista

                -->  EPS

else_blok       -->  ELSE_ jmpmodif NEWLINE
                     lista_naredbi

                -->  EPS
```

Prilikom prevođenja ključne riječi ELSEIF_ i ELSE_ zamijenjuju se bezuvjetnim skokovima na kraj strukture. Implicitni kratki oblici tih skokova mogu se modificirati modifikatorom dugog skoka, npr.:

```
    IF_
      ...
    IS_  EQUAL
      ...
    ELSEIF_  (L)
      ...
    IS_  TRUE
      ...
    ELSE_  (L)
      ...
    ENDIF_
```

3.4.5 LOOP_ struktura

Struktura petlje može imati tri osnovna oblika: sa izlazom na vrhu, na dnu, i bez izlaza na vrhu ili dnu. Sva tri oblika mogu imati i izlaze iz sredine.

3.4.5.1 Petlja sa izlazom na vrhu

Logički izraz koji predstavlja uvjet ponavljanja petlje nalazi se na njenom početku iza ključne riječi WHILE_, a kraj petlje označava ključna riječ REPEAT_:

```
    LOOP_  WHILE_  (logički_izraz)
             ...
    REPEAT_
```

Pojedine IS_ instrukcije u izrazu translatiraju se u uvjetne skokove na prvu instrukciju iza petlje, a instrukcija REPEAT_ u bezuvjetni skok na početak petlje. Taj skok može se modificirati navođenjem modifikatora dugog skoka iza ključne riječi REPEAT_.

3.4.5.2 Petlja sa izlazom na dnu

Sa ovim oblikom petlje već smo se susreli u primjerima. Logički izraz također slijedi riječ WHILE_, ali se nalazi na kraju petlje:

```
LOOP_
        ...
REPEAT_  WHILE_  (logički_izraz)
```

Pojedine IS_ instrukcije u izrazu translatiraju se u uvjetne skokove na početak petlje.

3.4.5.3 Petlja bez izlaza na vrhu ili dnu

Logički izraz ovdje se ne pojavljuje ni na početku ni na kraju petlje:

```
LOOP_
        ...
REPEAT_
```

Instrukcija REPEAT_ translatira se u bezuvjetni skok na početak petlje, koji se može modificirati modifikatorom dusos skoka.

3.4.5.4 Izlazi iz sredine petlje

Izlaze is sredine petlje mogu imati sva tri osnovna oblika petlje. Promatrano sa stanovišta logičkog srananja programa, potreba za njima obično se javlja samo kod trećeg oblika, a u ostalim oblicima izlazi iz sredine mogu pridonijeti efikasnosti. Izlaz iz sredine petlje specificira se primjenom ključne riječi WHILE_ koju slijedi logički izraz:

```
LOOP_
        (lista_naredbi)
WHILE_  (logički_izraz)
        (lista_naredbi)
REPEAT_
```

Pojedine IS_ instrukcije u logičkom izrazu prevode se u uvjetne skokove na prvu instrukciju iza petlje.
Ovakvih izlaza u jednoj petlji može biti proizvoljno mnogo, npr.:

```
LOOP_
        (lista_naredbi)
WHILE_  (logički_izraz)
        (lista_naredbi)
WHILE_  (logički_izraz)
        (lista_naredbi)
REPEAT_ WHILE_ (logički_izraz)
```

Činjenica da kod procesora MC68000 neizvršeni kratki skok traje manje od izvršenog može se iskoristiti za optimiziranje brzine petlji sa izlazom na dnu na račun memorijskog prostora /MORTON86/. To se postiže "odmatanjem" petlje, što u SYSTRAL-u omogućuju upravo izlazi iz sredine petlje.
Na primjer, petlju za traženje prve lokacije jednake nuli:

```
LOOP_
      TST.B  (A0)+
REPEAT_ WHILE_
IS_ NOT_ ZERO
```

možemo napisati i ovako, čime se postiže veća brzina:

```
LOOP_
      TST.B  (A0)+
WHILE_
IS_ NOT_ ZERO
      TST.B  (A0)+
REPEAT_ WHILE_
IS_ NOT_ ZERO
```

Brzinu možemo dalje povećavati dodajući nove parove TST.B - WHILE_.

Izlaze iz sredine petlje omogućuje sintaksni element (while_lista) sa ovakvom sintaksom:

```
while_lista   --)   WHILE_ logički_izraz
                    NEWLINE lista_naredbi
                    while_lista
              --)   EPS
```

3.4.5.5 Modifikator ulaska u petlju

U slučaju najjednostavnijeg logičkog izraza u svakom prolazu petlje sa izlazom na dnu izvršava se samo jedan skok, dok se kod petlje sa izlazom na vrhu u svakom prolazu osim eventualno u prvom izvršavaju dva skoka, što je manje efikasno.
Petlja sa izlazom na dnu može postati logički ekvivalentna petlji sa izlazom na vrhu, ali efikasnija od nje, primjenom modifikatora ulaska u petlju (-1) :

```
LOOP_  (-1)
          ...
REPEAT_ WHILE_
          ...
IS_  EQUAL
```

Ovaj modifikator generira bezuvjetni skok na odsječak iza ključne riječi REPEAT_. Taj skok izvršava se samo prilikom ulaska u petlju, a može se modificirati da postane dus :

```
(-1)  (L)
```

Smisao oznake -1 je u neizvršavanju tijela petlje u prvoj iteraciji.

Sintaksa modifikatora ulaska u petlju je slijedeća:

```
entrymodif      --)   MODIF_M  imnmodif
                --)   EPS
```

3.4.5.6 UNTIL_ - EVENT_ mehanizam

Dosad opisanim mehanizmima ne mogu se izraziti sve potrebe za izlazima iz petlje koje se mogu pojaviti - takav je slučaj kad se unutar petlje nalazi neka struktura i na nekom mjestu unutar te druge strukture je potrebno izvršiti izlazak iz petlje.

U svim dosad prikazanim oblicima strukture petlje vizualnim praćenjem strukture vrlo lako se mogu uočiti svi njeni izlazi jer su specifikacije izlaza uvijek smještene uz neku od granica strukture: gornju, donju ili lijevu. Sličan je pristup kod mehanizma za omogućavanje izlazaka iz drugih struktura unutar petlje: na početku petlje mora se deklarirati da postoje izlazi negdje na sredini. Deklaraciju obavlja naredba UNTIL_, a stvarni izlazak iz petlje provode EVENT_ naredbe.

Naredba

```
UNTIL_ (indikator_događaja)
```

može se pojaviti na početku svih oblika petlje. Element koji smo ovdje označili kao (indikator_događaja) zapravo je leksički element IDENT (identifikator). Smisao ovakve deklaracije je slijedeći: petlja se izvršava sve do pojave određene situacije, ili drugačije rečeno, sve dok se ne dogodi događaj čije ime označava (indikator_događaja).

Naredba EVENT_ ima dva oblika:

```
- EVENT_  (indikator_događaja)
- EVENT_  (indikator_događaja)  IF_
          (logički_izraz)
```

Prvi oblik izražava da se događaj dogodio i odmah provodi izlazak iz petlje, a drugi govori da se događaj dogodio ako je ispunjen uvjet logičkog izraza i u tom slučaju provodi izlazak. Prvi oblik translatira se u bezuvjetni skok koji se može modificirati, npr.:

```
EVENT_  FOUND  (L)
```

Naravno, (indikator_događaja) u EVENT_ naredbi mora biti jednak onome u pripadnoj UNTIL_ naredbi.

Ilustrirajmo primjenu UNTIL_ - EVENT_ mehanizma primjerom: treba ispisati datoteku na štampaču, ali tako da na početku i kraju svake stranice određeni broj linija ostane prazan. Pretpostavljamo da potprogram GETCHAR čita slijedeći znak datoteke sa standardnog ulaza u registar D0, a PRINT ispisuje znak iz registra D0 na štampač.

```
EMPTY EQU 3
PAGE  EQU 72 - 2 * EMPTY
CR    EQU 13
EOF   EQU -1

    MOVEM.L D0-D2,-(SP)    : spremi registre
    MOVE.B  # CR, D0_CR
!-------------------------------------------!
! LOOP_  UNTIL_  END_OF_FILE                !
!   !-------------------------------------!!
!   ! FOR_  D1_LINE, # EMPTY (-1) , #0    !!
!   !     PUTCHAR  D0_CR                  !!
!   ! NEXT_ D1_LINE                       !!
!   !-------------------------------------!!
!   !-------------------------------------!!
!   ! FOR_  D1_LINE, # PAGE (-1) , #0      !!
!   !   !---------------------------------!!!
!   !   ! LOOP_                           !!!
!   !   !     GETCHAR  D2_ZNAK            !!!
!   !   !   !-------------------------!!!!!
!   !   !   ! EVENT_  END_OF_FILE  IF_  !!!!!
!   !   !   !     CMP.B  # EOF, D2_ZNAK !!!!!
!   !   !   ! IS_ EQUAL                 !!!!!
!   !   !   !-------------------------!!!!!
!   !   !     PRINT  D2_ZNAK            !!!
!   !   ! REPEAT_ WHILE_                !!!
!   !   !     CMP.B  D0_CR, D2_ZNAK     !!!
!   !   ! IS_ NOT_ EQUAL                !!!
!   !   !---------------------------------!!!
!   ! NEXT_ D1_LINE                       !!
!   !-------------------------------------!!
!   !-------------------------------------!!
!   ! FOR_  D1_LINE, # EMPTY (-1) , #0    !!
!   !     PUTCHAR  D0_CR                  !!
!   ! NEXT_ D1_LINE                       !!
!   !-------------------------------------!!
! REPEAT_                                   !
!-------------------------------------------!
    MOVEM.L (SP)+,D0-D2    : obnovi registre
    RTS
```

Unutar petlje sa UNTIL_ deklaracijom može se nalaziti proizvoljno mnogo naredbi sa istim indikatorom događaja; drugim riječima, izlaza iz te petlje može biti proizvoljno mnogo.

Doseg deklaracije događaja, tj. područje u kojem je ona aktivna, proteže se od njene pojave na početku petlje do kraja petlje. Deklaracija može u dijelu svog dosega biti nevidljiva, ako je prekriva druga deklaracija istog identifikatora.

U primjeru

```
LOOP_  UNTIL_  FOUND
         ...
         EVENT_  FOUND
         ...
         LOOP_  UNTIL_  FOUND
                  ...
                  EVENT_  FOUND
         REPEAT_
         ...
REPEAT_
```

prva EVENT_ naredba odnosi se na vanjsku petlju, a druga na unutarnju.

Elementi UNTIL_ - EVENT_ mehanizma imaju slijedeću sintaksu:

```
until_blok      --)  UNTIL_  IDENT
                --)  EPS

event_naredba   --)  EVENT_  IDENT
                     event_ostatak  NEWLINE

event_ostatak   --)  jmpmodif
                --)  IF_ logički_izraz
```

3.4.5.7 Sintaksa LOOP_ strukture

Pošto smo opisali sve njene elemente, konačno možemo predstaviti sintaksu cijele LOOP_ strukture:

```
loop_struktura  --)  LOOP_  entrymodif
                     until_blok loop_ostatak
                     NEWLINE

loop_ostatak    --)  WHILE_  logički_izraz
                     NEWLINE lista_naredbi
                     while_lista  REPEAT_
                     jmpmodif

                --)  NEWLINE  lista_naredbi
                     while_lista  REPEAT_
                     loop_izlaz

loop_izlaz      --)  jmpmodif
                --)  WHILE_  logički_izraz
```

3.4.6 FOR_ struktura

Druga struktura petlje u SYSTRAL-u nije tako općenita kao analogne strukture u višim jezicima. Ulogu kontrolne varijable petlje ima neki od registara procesora, petlja uvijek napreduje u silaznom smjeru sa korakom -1, a krajnja vrijednost petlje je fiksno definirana (kod procesora MC68000 ta vrijednost je 0, a kod Z80 -1). Struktura se implementira pomoću instrukcija tipa "decrement and branch" (DBcc kod MC68000) ako procesor sadrži takve instrukcije, a inače instrukcijom za smanjivanje sadržaja registra i zatim uvjetnim skokom.

Struktura FOR_ - NEXT_ može imati tri osnovna oblika, analogno LOOP_ strukturi:

```
        FOR_ ...
               ...
        NEXT_ ...
```

```
        FOR_ ...  WHILE_  (logički_izraz)
               ...
        NEXT_ ...
```

```
FOR_ ...
        ...
NEXT_ ... WHILE_ (logički_izraz)
```

Kod prva dva oblika petlje naredba NEXT_ translatira se u DBRA instrukciju, a kod trećeg, u sprezi sa posljednjom IS_ instrukcijom u logičkom izrazu, u moćnu instrukciju DBcc.

Analogija sa LOOP_ strukturom ne prestaje kod osnovnih oblika petlje. Mogući su višestruki izlazi iz sredine petlje pomoću WHILE_ naredbi (sintaksni element (while_lista)) kao i izlazi iz struktura ugniježđenih u petlju, primjenom UNTIL_ - EVENT_ mehanizma; a može se primijeniti i modifikator ulaska u petlju.

3.4.6.1 Osnovni oblik FOR_ petlje

Jednostavni, najčešće upotrebljavani oblik FOR_ petlje izgleda ovako:

```
FOR_ (registar), (poč_vrij), (kraj_vrij)
            (lista_naredbi)
NEXT_ (registar)
```

U izvedbi za procesor MC68000 (registar) mora biti podatkovni registar, za početnu vrijednost ((poč_vrij)) su dozvoljeni svi adresni načini procesora, a krajnja vrijednost ((kraj_vrij)) je uvijek #0. Naravno, registar koji se pojavljuje u NEXT_ naredbi mora biti jednak registru iz FOR_ naredbe.

Kao primjer napišimo odsječak za zbrajanje 10 članova polja:

```
  CLR  D0_SUM
 !-----------------------------------!
 ! FOR_ D1_COUNT, # 9, # 0           !
 !       ADD  (A0_TABLE)+, D0_SUM    !
 ! NEXT_ D1_COUNT                    !
 !-----------------------------------!
```

Instrukcija FOR_ translatira se u instrukciju punjenja kontrolnog registra početnom vrijednošću, dakle

```
MOVE   # 9, D1  ;
```

a NEXT_ u DBRA instrukciju.
Početna vrijednost nije 10, već 9, pošto je krajnja vrijednost 0.

3.4.6.1.1 Izostanak specifikacije početne vrijednosti petlje

Početna vrijednost petlje ne mora se uopće specificirati, već se umjesto nje mogu navesti dvije uzastopne točke. Tada kompilator ne generira početnu instrukciju inicijalizacije registra, jer se smatra da je registar već inicijaliziran i sadrži ispravnu početnu vrijednost petlje. Potreba za ovakvim oblikom FOR_ naredbe može se javiti npr. u slučaju kad je kontrolni registar petlje argument potprograma u kom se petlja nalazi.
Transformirajmo prethodni primjer u potprogram sa argumentom u registru D1 koji predstavlja broj članova polja:

```
  CLR  D0_SUM
 !-----------------------------------!
 ! IF_                               !
 !       TST  D1_COUNT               !
 ! IS_ NOT_ ZERO                     !
 !       SUBQ  #1, D1_COUNT          !
 !     !-----------------------------!!
 !     ! FOR_ D1_COUNT, .. , #0      !!
 !     !       ADD (A0_TABLE)+, D0_SUM !!
 !     ! NEXT_ D1_COUNT              !!
 !     !-----------------------------!!
 ! ENDIF_                            !
 !-----------------------------------!
  RTS
```

Da bi potprogram bio općenit morali smo uzeti u obzir i slučaj kad je broj jednak 0. Tada se ne smije izvršiti nijedan prolaz kroz petlju, što osigurava IF_ struktura. Prije početka petlje broj treba smanjiti za 1.

3.4.6.1.2 Modifikator ulaska u petlju u FOR_ strukturi

Modifikator ulaska u petlju opisali smo u prikazu LOOP_ strukture. On omogućuje ukazak u petlju na njenom dnu umjesto na početku, a u FOR_ strukturi može se pojaviti iza specifikacije početne vrijednosti petlje, bilo da je ona puna ili prazna.

Slijedeći primjer je varijanta prethodna dva - broj članova polja određuje memorijska lokacija N:

```
  CLR  D0_SUM
 !-----------------------------------!
 ! FOR_ D1_COUNT, N (-1) , #0        !
 !         ADD  (A0_TABLE)+, D0_SUM  !
 ! NEXT_ D1_COUNT                    !
 !-----------------------------------!
```

Modifikator ulaska u petlju generira bezuvjetni skok na kraj petlje, dakle na DBRA instrukciju.
U FOR_ strukturi oznaka -1 izražava ne samo neizvršavanje tijela petlje u prvoj iteraciji, već i činjenicu da se prije prvog izvršavanja tijela petlje (ako do njega uopće dođe) početna vrijednost petlje smanjuje za 1.

3.4.6.1.3 Modifikatori početnog punjenja registra petlje

Početno punjenje kontrolnog registra petlje implicitno se ostvaruje MOVE instrukcijom. Međutim, ponekad mogu biti prikladnije instrukcije MOVE.B, MOVE.L ili MOVEQ i tada zahtjeve za generiranjem tih instrukcija izražavamo ubacivanjem modifikatora početnog punjenja registra petlje u izvorni tekst.

To su (B) , (L) i (Q) .

Navode se iza specifikacije početne vrijednosti petlje, npr.:

```
FOR_  D0 , (A1) (B) , #0
```

ili

```
FOR_  D2 , #7 (Q) , #0
```

a ako postoji i modifikator ulaska u petlju on se navodi iza njih.

Sintaksa je slijedeća:

```
loadmodif       -->     sizemodif
                -->     MODIF_Q

sizemodif       -->     MODIF_B
                -->     MODIF_L
```

Element (sizemodif) koristi se i u SWITCH_ strukturi.

3.4.6.2 FOR_ petlja sa logičkim izrazom na dnu

FOR_ petlju sa logičkim izrazom na vrhu ne treba posebno objašnjavati, pa zato prelazimo na treći oblik petlje:

```
FOR_ (registar), (poč_vrij), (kraj_vrij)
        (lista_naredbi)
NEXT_ (registar) WHILE_ (logički_izraz)
```

Osnovni razlog postojanja ovog oblika FOR_ strukture je omogućavanje generiranja instrukcije DBcc.

Riješimo na primjer zadatak pomicanja 6 nižih bitova registra, sve dok se ne pomakne prvi bit jednak nuli:

```
FOR_ D1, #5, #0
        LSR  #1, D2
NEXT_ D1 WHILE_
IS_ CARRY
```

Instrukcije NEXT_ i IS_ zajedno se translatiraju u instrukciju DBCC.

Ako je logički izraz složeniji generirani kod za pojedine IS_ instrukcije se razlikuje:

```
LAB1                        ; FOR_ D0, .. , #0
        ...
                            ; NEXT_ D0 WHILE_
        ...
        BEQ.S   LAB2        ; IS_ TRUE OR_
        DBRA    D0, LAB1
        BRA.S   LAB3
LAB2
        ...
        BEQ.S   LAB3        ; IS_ TRUE AND_
        ...
        DBEQ    D0, LAB1    ; IS_ TRUE
LAB3    ...
```

Instrukcija IS_ sa AND_ operatorom translatira se u uvjetni skok kojim se izlazi iz petlje, a ona sa OR_ operatorom u niz od tri instrukcije.

3.4.6.3 Sintaksa načina adresiranja procesora MC68000

Od sintaksnih elemenata koje ćemo ovdje prikazati, u FOR_ strukturi koriste se (argument) i (data_registar), a u SWITCH_ strukturi i (adresni_registar) te (memorija).

```
argument            -->   registar
                    -->   OZNAKA_BROJA
                          neposredni_argument
                    -->   memorija

registar            -->   data_registar
                    -->   adresni_registar

data_registar       -->   DATA_REG
                    -->   SUPST

adresni_registar    -->   ADR_REG
                    -->   MODIF_A  SUPST

adresni_pc_registar -->   adresni_registar
                    -->   PC_REG

neposredni_argument -->   NUM_KONST
                    -->   IDENT
                    -->   ZN_KONST
                    -->   SUPST

memorija            -->   L_ZAG  adresni_registar
                          D_ZAG  plus

                    -->   MINUS  L_ZAG
                          adresni_registar  D_ZAG

                    -->   vrijednost  ind_način

ind_način           -->   L_ZAG
                          adresni_pc_registar
                          drugi_registar  D_ZAG

                    -->   EPS

plus                -->   PLUS
                    -->   EPS

vrijednost          -->   NUM_KONST
                    -->   IDENT
                    -->   ZN_KONST

drugi_registar      -->   ZAREZ  registar
                    -->   EPS
```

Vidimo da se kao registar podataka, adresni registar i neposredni argument može pojaviti i supstitucija makro argumenta (SUPST). U slučaju adresnog registra elementu SUPST mora prethoditi modifikator adrese (A).

3.4.6.4 Sintaksa FOR_ strukture

Sintaksa cijele strukture je ovakva:

```
for_struktura   -->   FOR_  data_registar
                      ZAREZ  drugi_argument
                      entrymodif  ZAREZ
                      OZNAKA_BROJA  NUM_KONST
                      until_blok  for_ostatak

for_ostatak     -->   WHILE_  logički_izraz
                      NEWLINE  lista_naredbi
                      while_lista  NEXT_
                      data_registar

                -->   NEWLINE  lista_naredbi
                      while_lista  NEXT_
                      data_registar  for_izlaz

for_izlaz       -->   WHILE_  logički_izraz
                -->   EPS

drugi_argument  -->   argument  loadmodif
                -->   TOČKA  TOČKA
```

3.4.7 MONITOR_ struktura

Ova struktura omogućuje strukturirani zapis u onim slučajevima koji se ne mogu efikasno riješiti upotrebom prostalih struktura. Ona predstavlja proširenje koncepta UNTIL_ - EVENT_ za omogućavanje izlaza iz petlje sa bilo kojeg nivoa gniježđenja unutar nje.
Opći oblik strukture je slijedeći:

```
MONITOR_ (događaj_1), .. , (događaj_N)
        (lista_naredbi)
WHEN_  (događaj_1)
        (lista_naredbi_1)
    ...
WHEN_  (događaj_N)
        (lista_naredbi_N)
RESUME_
```

U naredbi MONITOR_ deklariraju se događaji koji se nadziru u listi naredbi koja slijedi. Unutar te liste naredbi nalaze se EVENT_ naredbe koje indiciraju pojavu pojedinih događaja, tj. prenose kontrolu na pripadne liste naredbi specificirane odgovarajućim WHEN_ naredbama. Kada se dogodi neki događaj,

nakon izvršavanja pridružene liste naredbi struktura završava, tj. kontrola se prenosi na instrukcije što je slijede. Ako se unutar prve liste naredbi ne dogodi nijedan događaj i kontrola dođe do prve WHEN_ naredbe, struktura odmah završava.

Za razliku od LOOP_ i FOR_ strukture u kojima se doseg deklaracije događaja proteže cijelom strukturom, doseg deklaracija u MONITOR_ naredbi proteže se od početka strukture do prve WHEN_ naredbe. Bilo koji događaj deklariran u MONITOR_, LOOP_ ili FOR_ naredbi može u dijelu svog dosega biti nevidljiv, tj. prekriven deklaracijom istog događaja u drugoj MONITOR_, LOOP_ ili FOR_ naredbi.

Broj i redoslijed WHEN_ naredbi mora biti jednak broju i redoslijedu događaja deklariranih u MONITOR_ naredbi.

Svaka WHEN_ naredba translatira se u bezuvjetni skok na kraj strukture, te u labelu - cilj skokova dobivenih translacijom EVENT_ naredbi. Skok koji generira WHEN_ naredba može se modificirati, npr.:

```
WHEN_  SUCCESS  (L)
```

Međutim, u nekim slučajevima taj skok je nepotreban (npr. kad je lista naredbi pridružena zadnjoj WHEN_ naredbi prazna). Upotreba modifikatora adrese u WHEN_ naredbi, npr.:

```
WHEN_  SUCCESS  (A)
```

uzrokuje samo generiranje labele (simboličke adrese), a ne i skoka.

Za ilustraciju dajemo program za binarno pretraživanje tabele, prikazan na slici 1. Baziran je na primjeru iz reference /HARMAN85/. Početna adresa tabele je u registru A0, u D1 je broj članova tabele, a u D2 broj bytova koje sadrži svaki član. Prvi byte svakog člana je ključ koji se uspoređuje sa zadanim ključem u D0. U registru A6 vraća se adresa člana tabele gdje je nađen ključ, ili 0 ako nije nađen.

```
MOVEM.L D2-D5,-(SP)  ; spremi registre
SUBQ.W  #1, D2_END   ; end = broj članova - 1
CLR.L   D3_BEGIN     ; begin = 0
MOVE.L  D3_BEGIN, A6_ADR_KEY  ; izlazni registar = 0
!-------------------------------------------------------!
! MONITOR_  FOUND, NOT_FOUND                            !
!       !-----------------------------------------------!!
!       ! LOOP_                                         !!
!       !       !---------------------------------------!!!
!       !       ! EVENT_  NOT_FOUND  IF_                !!!
!       !       !         CMP.W  D2_END, D3_BEGIN       !!!
!       !       ! IS_  GREATER                          !!!
!       !       !---------------------------------------!!!
!       !       !    ; ako je begin > end pretraživanje !!
!       !       !    ; završava neuspješno              !!
!       !       !  MOVE.W  D3_BEGIN, D4_MIDDLE          !!
!       !       !  ADD.W   D2_END, D4_MIDDLE            !!
!       !       !  LSR.W   #1, D4_MIDDLE                !!
!       !       !    ; middle = begin + end / 2         !!
!       !       !  MOVE.W  D4_MIDDLE, D5_OFFSET         !!
!       !       !  MULU    D1_LENGTH, D5_OFFSET         !!
!       !       !    ; offset = middle * length         !!
!       !       !---------------------------------------!!!
!       !       ! EVENT_  FOUND  IF_                    !!!
!       !       !         CMP.B 0(A0_TABLE,D5_OFFSET),D0_KEY !!!
!       !       ! IS_  EQUAL                            !!!
!       !       !---------------------------------------!!!
!       !       !    ; ako je ključ jednak vrijednosti u!!
!       !       !    ; tabeli na mjestu middle          !!
!       !       !    ; pretraživanje je uspješno završeno!!
!       !       !---------------------------------------!!!
!       !       ! IF_                                   !!!
!       !       ! IS_  LESS   ; ako je manji           !!!
!       !       !         SUBQ.W  #1, D4_MIDDLE         !!!
!       !       !         MOVE.W  D4_MIDDLE, D2_END     !!!
!       !       !           ; end = middle - 1          !!!
!       !       !                                       !!!
!       !       ! ELSE_    ; ako je veći                !!!
!       !       !         ADDQ.W  #1, D4_MIDDLE         !!!
!       !       !         MOVE.W  D4_MIDDLE, D3_BEGIN   !!!
!       !       !           ; begin = middle + 1        !!!
!       !       ! ENDIF_                                !!!
!       !       !---------------------------------------!!!
!       ! REPEAT_                                       !!
!       !-----------------------------------------------!!
! WHEN_  FOUND  (A)                                     !
!         LEA  0(A0_TABLE, D5_OFFSET), A6_ADR_KEY       !
!           ; upiši u izlazni registar adresu           !
!           ; gdje je nađen ključ                       !
! WHEN_  NOT_FOUND  (A)                                 !
! RESUME_                                               !
!-------------------------------------------------------!
MOVEM.L  (SP)+, D2-D5   ; obnovi registre
RTS
```

Slika 1. Program za binarno pretraživanje tabele

Sintaksa strukture je ovakva:

```
monitor_struktura  -->  MONITOR_ lista_ident
                        NEWLINE lista_naredbi
                        when_lista RESUME_
                        NEWLINE

lista_ident        -->  IDENT ost_ident

ost_ident          -->  ZAREZ lista_ident
                   -->  EPS

when_lista         -->  WHEN_ IDENT whenmodif
                        NEWLINE lista_naredbi
                        when_lista

                   -->  EPS

whenmodif          -->  MODIF_L
                   -->  MODIF_A
                   -->  EPS
```

3.4.8 SWITCH_ struktura

SWITCH_ struktura omogućuje selekciju (najviše) jedne od mnogih mogućih akcija u zavisnosti o vrijednosti selektorskog argumenta. Svakoj akciji pridružen je skup unaprijed definiranih, u načelu konstantnih vrijednosti. Ukoliko je neka od tih vrijednosti jednaka vrijednosti selektorskog argumenta izvodi se pripadna akcija i zatim struktura završava. U suprotnom se izvodi akcija adresirana ključnom riječi ELSE_, a ako ona ne postoji struktura odmah završava.

SWITCH_ struktura može se implementirati na više načina: pomoću CMP i TST instrukcija, pomoću indirektnog skoka preko tablice adresa, a uz neka ograničenja i pomoću DBRA instrukcije. Pristup SYSTRAL-a je podržavanje svih triju načina pri čemu programer u izvornom programu specificira željeni način. Implementacija indirektnim skokom moguća je samo kod dvoprolaznog kompilatora, tako da je definirana samo na drugom nivou jezika. Razvoj SYSTRAL-a nije potpuno završen. Trenutno je potpuno definiran samo prvi oblik SWITCH_ strukture i zato će jedino on biti ovdje opisan.

Opći oblik strukture je ovakav:

```
SWITCH_   (selektorski_argument)
          (lista_naredbi)
CASE_     (lista_argumenata)
          (lista_naredbi)
     ...
CASE_     (lista_argumenata)
          (lista_naredbi)
ELSE_
          (lista_naredbi)
ENDSWITCH_
```

Selektorski argument može biti registar podataka, adresni registar ili memorijska lokacija, i ovisno o tome struktura se implementira pomoću CMP, CMPA ili CMPI instrukcija, a ako ga slijedi modifikator veličine instrukcije dobivaju ekstenziju .B ili .L .
Prva lista naredbi, ako nije prazna, služi za određivanje vrijednosti selektorskog argumenta.
U strukturi mora postojati najmanje jedna CASE_ naredba. Osim prve, sve ostale ključne riječi CASE_, kao i ELSE_ naredba, translatiraju se u bezuvjetne skokove na kraj strukture i stoga ih mogu slijediti modifikatori dugog skoka.
Kao argumenti CASE_ naredbi dozvoljeni su svi adresni načini procesora MC68000. Pojedini argumenti u listi odvojeni su zarezima. Za svaki argument generira se CMP, CMPA, CMPI ili TST instrukcija, te uvjetni skok (može se modificirati modifikator smješten iza argumenta). Cilj skoka za zadnji argument u listi je slijedeća CASE_ ili ELSE_ naredba ili kraj strukture, a za zadnji argument početak pripadne liste naredbi.

Za primjer kodirajmo zadatak ispitivanja da li je slovo u registru D0 samoglasnik. Izlazna informacija ostavlja se u registru D1:

```
SWITCH_  D0_ZNAK (B)

CASE_ # 'A', # 'E', # 'I', # 'O', # 'U'

        MOVE  #1, D1_SAMOGLASNIK
ELSE_
        MOVE  #0, D1_SAMOGLASNIK
ENDSWITCH_
```

Na kraju prikazujemo sintaksu strukture:

```
switch_struktura  -->  SWITCH_ sw_argument
                       sizemodif NEWLINE
                       lista_naredbi CASE_
                       lista_argumenata
                       NEWLINE lista_naredbi
                       case_lista sw_else_blok
                       ENDSWITCH_ NEWLINE

sw_argument       -->  registar
                  -->  memorija

lista_argumenata  -->  argument jmpmodif
                       ostali_argumenti

ostali_argumenti  -->  ZAREZ lista_argumenata
                  -->  EPS

case_lista        -->  CASE_ jmpmodif
                       lista_argumenata
                       NEWLINE lista_naredbi
                       case_lista

                  -->  EPS

sw_else_blok      -->  ELSE_ jmpmodif
                       NEWLINE lista_naredbi

                  -->  EPS
```

3.4.9 Optimizacija kvalitete koda

U prvoj fazi prevođenja svaka struktura se prevodi zasebno, neovisno o kontekstu u kojem je smještena, zbog čega dobiveni kod nije uvijek optimalan. Kasnija faza optimizacije može poboljšati kvalitetu koda. Osnovna potrebna optimizacija je optimizacija lanaca skokova. Takav lanac čini npr. skok na mjesto gdje se nalazi drugi bezuvjetni skok, gdje se eventualno može nalaziti drugi bezuvjetni skok, itd., kao u slijedećem primjeru:

```
LOOP_
        ...
        IF_
                ...
        IS_ ZERO
                ...
        ENDIF_
REPEAT_
```

Lanac započinje uvjetnim ili bezuvjetnim skokom, DBRA ili DBcc instrukcijom. Optimizacija čini ciljeve svih skokova u lancu jednakim cilju posljednjeg skoka.

Ostale potrebne optimizacije zahtijevaju postojanje posebnih sintaksnih menanizama, u ovom času nepotpuno definiranih.

3.4.10 Opcije kompilatora

U naredbi za aktiviranje kompilatora posebno su korisne opcije -l i -s koje specificiraju da će svi generirani skokovi biti dugi, odnosno kratki, bez obzira na postojanje modifikatora. Prilikom razvoja programa, kad se on često mijenja, kompiliramo program uz opciju -l: konačnu verziju kompiliramo sa -s da vidimo gdje su potrebni dugi skokovi: tada na ta mjesta ubacimo modifikatore i kompiliramo bez opcija.

4. ZAKLJUČAK

U članku su opisane strukture i mehanizmi simboličkog strukturiranog zbirnog jezika SYSTRAL. Razvoj jezika nije potpuno dovršen.

U toku protekle godine članovi ekipe projekta digitalne regulacije električkih strojeva u Elektrotehničkom institutu "Rade Končar" koristili su SYSTRAL za razvoj programske podrške. Osnovni zahtjev postavljen na programsku podršku tog projekta je vrlo velika brzina rada. Zahtjev je zadovoljen jer je prilikom razvoja jezika i odabira njegovih struktura i mehanizama maksimalna pažnja posvećena efikasnosti.

5. ZAHVALE

Posebno zahvaljujem T. Crnogiji, prvom korisniku SYSTRAL-a, čija su iskustva i primjedbe pridonijele povećanju mogućnosti jezika i razlučivanju bitnih i manje bitnih aspekata. Zahvaljujem i Ž. Pelešu i I. Šumisi čije su primjedbe također pridonijele poboljšanju jezika, te N. Periću na podršci pruženoj razvoju ovog projekta.

6. LITERATURA

/KNUTH74/ D.E.Knuth: "Structured programming with go to statements", Computing Surveys, 6, 4, 261-301 (1974)

/COHEN83/ A.Cohen: "Structure, Logic, and Program Design", John Wiley, 1983.

/ZAHN74/ C.T.Zahn: "A control statement for natural top-down structured programming", Symposium on Programming Languages, Paris, 1974.

/WALKER81A/ G.Walker: "Toward a Structured 6809 Assembly Language, Part 1: An Introduction to Structured Assembly Language", BYTE, November 1981, 370-382

/WALKER81B/ G.Walker: "Toward a Structured 6809 Assembly Language, Part 2: Implementing a Structured Assembler", BYTE, December 1981, 198-228

/KRIEGER80/ M.Krieger: "Structured assembly language suits programmers and microprocessors", Electronics, January 17, 1980, 63-71

/MOSAK82/ A.Mosak: "Structured Programming Can be Applied to Microprocessors - Even by Novices (A Review of Structured Microprocessor Programming)", IEEE MICRO, February 1982.

/WIRTH68/ N.Wirth: "PL360, A Programming Language for the 360 Computers", Journal of the ACM, Vol 15, No.1, January 1968, 37-74

/KAWAI80/ S.Kawai: "A Semiblock Structure for Low-level Languages", Software - Practice and experience, vol. 10, 11-19 (1980)

/SMITH85/ M.F.Smith, Y.Hoffner, M.A. Sealey: "Mapping High - Level Syntax and Structure Into Assembly Language", IEEE MICRO, August 1985, 67-81

/ANDERSON81/ A.Anderson, M.Tracy, P. Wasson: "FORTH-79 Tutorial and Reference manual, Apple II Version, Volume I, Appendix II: 6502 Assembler", MicroMotion, 1981.

/HARMAN85/ L.Harman, B.Lawson: "The Motorola MC68000 Microprocessor Family: Assembly Language, Interface Design, and System Design", Prentice-Hall, 1985.

/MORTON86/ M.Morton: "68000 Tricks and Traps", BYTE, September 1986, 163-172

/WAITE84/ W.M.Waite, G.Goos: "Compiler Construction", Springer Verlag, 1984.

/NEŽIĆ85/ H.Nežić: "Strukturirano programiranje u asembleru", Informatica, 1/1985, 52-60

Matjaž Debevc
Metka Zorič
Rajko Svečko
Dali Đonlagič
Tehniška fakulteta Maribor

UDK 681.325

POVZETEK

Na področju računalništva prihaja vedno bolj do izraza področje računalniške grafike. Osebni računalniki postajajo tudi vedno bolj pristopni širokim množicam uporabnikov. Trenutno najbolj aktualna možnost grafičnega prikazovanja nam podaja EGA grafična kartica za IBM in kompatibilne osebne računalnike. Delo opisuje zgradbo, osnovo in programiranje te kartice.

ABSTRACT

Computer graphics is becoming more and more districtive in the field of computers. Personal computers are becoming acessible to a wide number of users. At the moment the EGA graphics card presents the best possibility of graphic display for IBM and a compatible personal computers. The present thesis discribes the structure and programming of this card.

1. UVOD

Računalniška grafika je ena najbolj spektakularnih možnosti, ki nam jih dajejo računalniki. Računalniška grafika je poseben medij, ki omogoča najlažjo in najhitrejšo komunikacijo med človekom in strojem. Človeško oko lahko dosti hitreje razbere informacijo z grafične slike, kakor pa s tabele, polne številk.

Dolga leta je bilo to področje izredno drago in težko dostopno. Zadnjih 30 let je čutiti enakomeren padec svetovnih cen na področju računalništva. Ta padec se giblje za okoli 15% letno. To je vzrok, da postajajo računalniki z velikimi zmogljivostmi vedno bolj dostopni. Povečanje zmogljivosti in padec cen se med drugim tudi čuti na področju osebnih računalnikov IBM in njemu kompatibilnih. Nekdaj niti pomisliti ni bilo mogoče na visoko zmogljivo grafiko. Slika, ki je bila nekoč narisana v nekaj minutah z majhno resolucijo zaslona, se danes nariše v trenutku na visoko resolucijskem ekranu. Ni daleč čas, ko bo trodimenzionalna grafika čisto nekaj vsakdanjega.

2. IBM – Enhanced Graphics Adapter

Na osebnih računalnikih IBM in njemu kompatibilnih je danes najbolj aktualna grafična kartica EGA, ki je našla pot v marsikateri osebni računalnik in je največkrat obravnavana kot standard za ostale grafične kartice. EGA kartico imenujejo tudi IBM-HR (High Resolution) kartica. To je kartica, ki postavlja nove možnosti na področju grafike na osebnih računalnikih.

Že več kot dve leti je ta kartica na trgu in že obstaja cela vsta tako imenovanih kompatibilnih EGA kartic. V grafičnem načinu je EGA kartica z resolucijo 640 * 350 točk za 150 grafičnih vrstic boljša od stare barvne grafične kartice CGA (Colour Graphic Adapter).

Takšna resolucija nam omogoča ukinitev utripajoče slike, ne glede na to ali uporabljamo monohromatski zaslon ali novi IBM "Enhanced Color Display". Seveda omogoča EGA kartica tudi emuliranje starega zaslona s resolucijo 320 * 200 ali 640 * 200 (glej Načini EGA kartice).

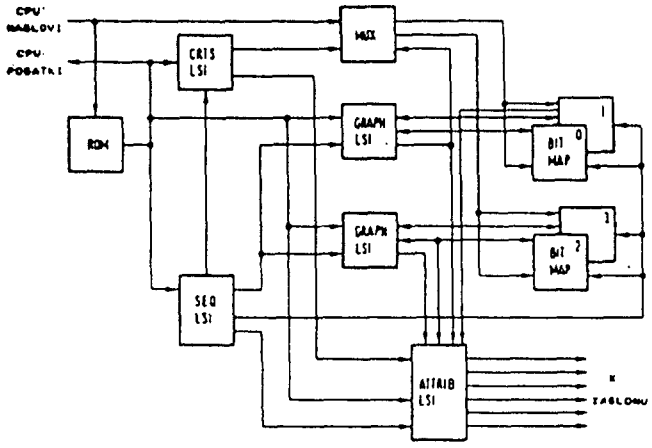
Za individualno uporabo EGA kartice sedaj ne obstaja veliko uporabniških programskih paketov. Družba Borland International

predstavlja s svojim TURBO PASCAL-om verzija 4.0 zelo dober in tudi poceni programski paket, ki med drugim podpira tudi EGA kartico. Vendar običajno tako velikega programskega paketa niti ne potrebujemo. Poleg tega mora biti program napisan v pascalu.

Za pisanje samo nujno potrebnih, kratkih funkcij za uporabo EGA grafične kartice je potrebno poznati zgradbo in delovanje grafične kartice.

2.1. ZGRADBA EGA KARTICE

Programiranje za EGA kartico je težko že zaradi množice registrov (70), kakor tudi zaradi množice funkcij, ki jih ta kartica omogoča. Že ko pogledamo v "Technical Reference Manual" in blokovni diagram za EGA kartico (slika 1), nam postane jasno, da je potrebno natančno razumevanje funkcij in registrov, če želimo, da bomo imeli pravilno delujoč programski paket. Kot osnova nam služi knjiga "IBM - Personal Computer, IBM - Enhanced Graphics Adapter".



slika 1 : Blokovni diagram EGA kartice

Glede na blokovni diagram delimo EGA kartico na:

2.1.1. CRT Controller

CRTC (Cathode Ray Tube Controller) je po funkciji razširjeni 6845. Generira sinhronizirane signale za horizontalne in vertikalne premike. Odgovarjajoči parametri se programirajo v 19 CRT registrih. Vertikalni registri so dolgi 9 bitov, kar nam da resolucijo 512 slikovnih točk. Če je drugi bit v "Mode Control Register" enak 1, se lahko resolucija podvoji na 1024 vrstic na sliko, kar pa pomeni, da potrebujemo poseben monitor. Nadalje skrbi CRTC za obnovitev slike shranjene v dinamičnem RAM-u. Naslov za obnovitev slike je dolg dva byta, kar pomeni, da CRTC dovoljuje naslavljanje do 64 Kbytov.

CRT kontroler dosežemo z vhodno/izhodnim naslovom 3?4 in 3?5, kjer "?" pomeni B ali D.B uporabimo za Monochrome način, D pa za barvni način. 3?4 je naslovni register, medtem ko je 3?5 bralno pisalni podatkovni register.
Vsebina registrov:

1. Address Register
2. HorizontalTotal
3. Horizontal Display End
4. Start Horizontal Blank
5. End Horizontal Blank
6. Start Horizontal Retrace
7. End Horizontal Retrace
8. Vertical Total
9. Overflow
10. Preset Row Scan
11. Max Scan Line
12. Cursor Start
13. Cursor End
14. Start Address High
15. Start Address Low
16. Cursor Location High
17. Cursor Location Low
18. Vertical Retrace Start
19. Light Pen High
20. Vertical Retrace End
21. Light Pen Low
22. Vertical Display End
23. Offset
24. Underline Location
25. Start Vertical Blank
26. End Vertical Blank
27. Mode Control
28. Line Compare

2.1.2. Zaslonski pomnilnik

Zaslonski pomnilnik se sestoji iz štirih pomnilniških prostorov (Bit Planes). V osnovni verziji je vsak prostor velik 16 Kbytov in se nahaja na EGA kartici. V dveh korakih se lahko kapaciteta dvakrat poveča. Potem govorimo o 64, 128 ali 256 Kbytni EGA kartici. Slika z resolucijo 640 * 350 ima 80 * 350 bytov. Organizacija slikovne točke v zaslonskem pomnilniku je v grafičnem načinu naslednja:

Prvih osem slikovnih točk zaslona (zgornji levi kot !) se nahaja na naslovu 000H slikovnega pomnilnika, naslednjih osem na naslovu 001H itd. Prva slikovna točka znotraj enega byta je bit 7, naslednja je bit 6.

2.1.3. Grafični in tekstovni način

Štirje pomnilniški prostori zaslonskega pomnilnika reprezentirajo v grafičnem načinu tudi naslov barvnih atributov ene slikovne točke (16 barv). Ta naslov je označen v Attribute-Controller-ju kot indeks v barvnem registru. Več o tem je napisano v

poglavju o Attribut-Controller-ju. Če ima EGA kartica 64 Kbytov na voljo, lahko uporabimo samo 4 barve.

V tekstovnem načinu je pomnilniški prostor znakovno kodni pomnilnik, pomnilniški prostor 1 je pomnilnik za atribute, pomnilniški prostor 2 pa je znakovni generator.

Če se selektira tekstovni način, prenese BIOS enega od dveh znakovnih stavkov v pomnilniški prostor 2.

2.1.4. Sekvenčni generator in multiplexer

Naslovni multiplexer (MUX) krmili naslove zaslonskega pomnilnika, ki pridejo enkrat direktno od CPU-ja (naložitev zaslonskega pomnilnika) in drugič od CRTC (zaslonska osvežitev). Sekvenčni generator (SEQ) generira krmilne signale za zaslonski pomnilnik in takt za zaslonsko ponavljanje. Map-Mask-Register omogoča individualno vpisovanje na zaslonski pomnilnik. Zraven lahko tudi izberemo določeno barvo. Za tekstovni način (A/N) se nahaja v Character-Map-Select registru izbira štirih znakovnih stavkov.

Sekvenčni register dosežemo z naslovi 3C4 (select naslova) in 3C5 (podatkovni vhodno - izhodni register).

Vsebina registra :

1. Naslov
2. Reset
3. Clocking Mode
4. Map Mask
5. Character Map Select
6. Memory Mode

2.1.5. Grafični kontroler

Grafični kontroler lahko dela v dveh načinih; v grafičnem ali v tekstovnem. V grafičnem načinu se prenesejo podatki serijsko preko tako imenovanih Bit-Plane vodil C0, C1, C2, C3 k Attribute - Controler-ju. Ta štiri vodila, vsako za zaslonski pomnilniški prostor (Bit Plane Map) tvorijo 16 možnih barv (glej Attribute Controller).

Prvi grafični kontroler je za zaslonski pomnilniški prostor 0 in 1 (2 byta) , drugi grafični kontroler pa za ostala dva (2 in 3). Oba grafična kontrolerja imata vsak po 16 bitni podatkovni register, kjer se lahko vpiše ali bere 32 bitov slikovnih podatkov v enem pomnilniškem ciklusu. Nadalje se nahaja v obeh grafičnih kotrolerjih še krmiljenje za izbiro slikovne točke in njegove barve, kakor tudi štiri logične funkcije (zamenjava, AND, OR, XOR).

Color-Compare register omogoča hkratno primerjanje z eno določeno barvo preko osmih slikovnih točk enega byta. V tekstovnem načinu se podatki transportirajo direktno v Attribute-Controller.

Register dosežemo z naslovi 3CE (select naslova) in 3CF (podatkovni vhodno izhodni naslov).

Vsebina registra:

1. Graphics 1 Position
2. Graphics 2 Position
3. Graphics 1 & 2 naslov
4. Set/Reset
5. Enable Set/Reset
6. Color Compare
7. Data Rotate
8. Read Map Select
9. Mode Register
10. Miscellaneous
11. Color Don't Care
12. Bit Mask

2.1.6. Attribut Controller

Ta kontroler je zadnja postaja slikovnih podatkov na poti k zaslonu. Ima 21 registrov. Prvih 16 naslovov so tako imenovani barvno paletni registri. Biti 0 do bita 5 teh 16 registrov omogočajo dinamično izbiro 64 možnih barv.

Barvno paletni register se naslavlja preko štirih vodil C0 - C3, ki pridejo od grafičnega kontrolerja. To se dogaja serijsko, torej točka za točko. Preden enega od teh 16 registrov naslavljamo, gredo podatki tudi preko Color-Plane-Enable registra. Ponavadi se naslov pusti tukaj na miru, da imamo lahko na voljo 16 barv. Z Color-Plane-Enable registrom pa lahko omogočamo prikaz enega od štirih slik, ali pa kombinacijo teh štirih slik.

V tekstovnem načinu se tekstovni podatki prenašajo paralelno iz slikovnega pomnilnika v Attribut Controller.

Osvežilni naslov generiran od CRTC se naslavlja najprej na slikovni pomnilniški prostor 0 in 1.

Byte naslova, shranjen v pomnilniškem prostoru 1 se prenese v Attribut Controller, kjer ima za vsakokratno prikazano črko svojo veljavnost. Prebrana znakovna koda iz pomnilniškega prostora 0, na primer 41H za črko "A", se naslavlja v povezavi z Row-Scan-Count registrom zaslonskega pomnilniškega prostora 2, kjer se nahaja ta v ROM-u naložen znak.

Prebrani biti se pošljejo skozi grafični kontroler k Attribut kontrolerju, skupaj z barvnimi vrednostmi in so nato vodeni serijsko dalje k zaslonu. Row-Scan-Count register nastavi vertikalno višino znaka na zaslonu, to pomeni število horizontalnih vrstic iz katerih je znak sestavljen. Znak je lahko velik maximalno

32 znakov in se nastavi v CRTC registru 9.

Vsebina:

1. Address Register
2. Palette Register
3. Mode Control Register
4. Overscan Color Register
5. Color Plane Enable Register
6. Horizontal Pel Panning Register

3. PROGRAMIRANJE EGA KARTICE

Prejšne poglavje je bilo namenjeno spoznavanju zgradbe in strukture EGA kartice. To poglavje pa je namenjeno programiranju EGA kartice.

Namesto da bi pisali velik programski grafični paket, včasih potrebujemo samo osnovne ukaze za risanje na ekran, kakor so na primer vklop, izklop grafike, risanje črte in brisanje zaslona. V resnici obstaja veliko programskih paketov, ki podpira grafiko na EGA kartici za IBM-PC in kompatibilne, vendar programerji, ki želijo imeti hitre in učinkovite programe raje sežejo po osnovnih funkcijah, ki jih lahko sami napišejo.

Za programiranje EGA kartice potrebujemo EGA-BIOS funkcije. V ROM-u kartice se nahajajo poleg teh funkcij še oba znakovna stavka in ena samostojna funkcija.

EGA-BIOS vsebuje interrupt (prekinitvene) vektorje 05H in 10H. (H pomeni heksadecimalno). Interrupt 05H uporabimo za izpis na zaslon, medtem ko uporabljamo interrupt 10H za 19 krmilnih funkcij EGA kartice.

Ker se na PC-ju največkrat uporabljata prevajalnika TURBO-PASCAL in TURBO-C od firme BORLAND, so tudi podrogrami napisani v teh dveh jezikih. Prenos na prevajalnike ostalih firm ni problematičen, samo potrebno je vedeti, kako kličemo BIOS funkcije. Programa vključujeta ukaze za vklop, izklop grafike, risanje točke (pixla) in brisanje ekrana.

3.1. Nastavitev načina EGA kartice

Kadar želimo risati z EGA kartico, moramo najprej vklopiti EGA v ustrezni način.

Delovni načini EGA kartice:

EGA način	resolucija	število barv
C 0	320 X 200	16
E 0	320 X 350	16/64
C 1	320 X 200	16
E 1	320 X 350	16/64
C 2	640 X 200	16
E 2	640 X 350	16/64
C 3	640 X 200	16
E 3	640 X 350	16/64
C 4	320 X 200	4
C 5	320 X 200	4
C 6	640 X 200	2
M 7	720 X 350	4
C D	320 X 200	16
M E	640 X 200	16
M F	640 X 350	4
M 10	640 X 350	4/16

C - IBM Color zaslon, E - IBM Enhanced Color
M - IBM Monochrom zaslon

3.1.1. Za vklop in izklop grafičnega zaslona uporabljamo naslednje BIOS funkcije:

AH = 0; izbira BIOS načina
AL: EGA način po zgornji tabeli

AH = 0FH; prikaz trenut. zaslonskega statusa
parametri, ki vračajo podatke so:
AL: EGA - način
AH: število znakov/vrstico
BH: število trenutnih zaslonskih strani

Podprograma:

Če uporabljamo verzijo 4.0 TURBO-PASCALA, potem v glavi programa vstavimo :

```
uses DOS;
```

tip pa je definiran z Registers.

Kadar pa uporabljamo verzijo 3.0 pa je potrebno dodati :

```
type Registers =
        record
          case integer of
            1: (ax,bx,cx,dx,bp,si,di,ds,
                es,flgs: integer);
            2: (al,ah,bl,bh,cl,c,dl,dh: byte);
        end;

procedure Grafika(vklop: boolean);
var
  reg: Registers;
  nacin: integer;

begin {Grafika}
  if vklop then
    begin
      { branje aktualne nastavitve zaslona }
      reg.ax:= $0F00;
      Intr($10,reg);
      { shranitev načina }
      nacin:= reg.al;

      { EGA način za Enhanced Color Display }
      reg.ax:= $0010;
      Intr($10,reg);

    end
  else
    begin
      reg.ah:= 0;  { izklop grafičnega načina }
      reg.ax:= nacin;
      Intr($10,reg);
    end;
end; {Grafika}
```

Jezik C:

```
# include <dos.h>

union REGS reg;
int nacin;

Grafika(vklop)
int vklop;

{
  if (vklop)
  {
    reg.x.ax = 0x0F00;
    int86(0x10,&reg,&reg);

    reg.x.ax = 0x0010;
    int86(0x10,&reg,&reg);
  }
  else
  {
    reg.h.ah = 0;
    reg.x.ax = nacin;
    int86(0x10,&reg,&reg);
  }
}
```

3.1.2. Za risanje točke uporabljamo naslednje BIOS funkcije:

```
AH = 0CH;       nariši grafično točko
     BH:          tekoča stran
     DX:          Y pozicija (0 - 349)
     CX:          X pozicija (0 - 639)
     AL:          barva (0 - 63)


AH = 0DH;       beri grafično točko
     BH:          tekoča stran
     DX:          Y pozicija (0 - 349)
     CX:          X pozicija (0 - 639)
Vrnjen parameter:
     AL:          barva željene vrednosti
```

Podrogram za risanje točke v pascalu:

```
procedure Tocka(x,y,barva);
begin {Tocka}
  reg.ah:= $0C;     { riši grafično točko }
  reg.cx:= x;
  reg.dx:= y;
  reg.al:= barva;
  Intr($10,reg);
end; {Tocka}
```

Program v C jeziku:

```
Tocka(x,y,barva)
int x,y,barva
{
  reg.x.ah = 0x0C;
  reg.x.cx = x;
  reg.x.dx = y;
  reg.h.al = barva;
  int86(0x10, &reg, &reg);
}
```

3.1.3. Za brisanje ekrana uporabimo naslednjo BIOS funkcijo:

```
AH = 07H;  pomik trenutne strani navzdol
    AL:       vstavitev okna
    CH:       spodnji levi rob (vrstica)
    CL:       spodnji levi rob (kolona)
    DH:       zgornji desni rob (vrstica)
    DL:       zgornji desni rob (kolona)
    BH:       barva ozadja;
```

Podprogram za brisanje ekrana v pascalu:

```
procedure brisiZaslon;
begin {brisiZaslon}
  reg.ah:= $07;          { pomik strani navzdol }
  reg.al:= 0;            { vstavitev okna }
  reg.ch:= 0;            { spodnji levi rob }
  reg.cl:= 0;
  reg.dh:= 24;           { zgornji desni rob }
  reg.dl:= 79;
  reg.bh:= 0;            { barva ozadja }
  Intr($10,reg);
end; {brisiZaslon}
```

V C jeziku :

```
brisiZaslon()
{
   reg.h.ah = 0x07;
   reg.h.al = 0;
   reg.h.ch = 0;
   reg.h.cl = 0;
   reg.h.dh = 24;
   reg.h.dl = 79;
   reg.h.bh = 0;
   int86(0x10, &reg, &reg);
}
```

LITERATURA

Johnson Nelson:
ADVANCED GRAPHICS IN C
Programming and Techniques,

EGA Users' Manual

IBM Personal Computer:
IBM Enhanced Graphics Adapter,

IBM Personal Computer:
TURBO PASCAL V4.0.

Matjaž Debevc
Rajko Svečko
Mark Martinec*
Tehniška fakulteta Maribor
* Institut »Jožef Stefan«

UDK 681.326

POVZETEK

V želji, da bi uporabili Partner kot samostojen grafični terminal, je bil napisan grafični protokol med njim in med računalnikom VAX. S tem smo dosegli, da se uporabljajo vsi Partnerjevi grafični ukazi. Pri tem smo uporabili pomožni knjižnici TEKUSR na Partnerju in SUNLET na VAX sistemu.

ABSTRACT

In wish that we use Partner as an indepandet graphic terminal, was written a special graphic protocol between Partner and Vax. With that we are able to use all Partner's graphics instructions.
We used additional libraries: TEKUSR on Partner and SUNLET on VAX.

1. UVOD

Zaradi raznovrstnih možnosti uporabe vseh mogočih grafičnih paketov, ki so namenjeni velikim, dragim in težko dostopnim tujim grafičnim terminalom bi bilo ugodno, ko bi lahko vse te pakete uporabljali kar na mikroračunalnikih - sadovih naše, jugoslovanske ustvarjalnosti.

Primer takega mikroračunalnika je Iskra-Delta Partner z grafično kartico. Mikroračunalnik Partner je samostojna enota z enobarvnim, grafičnim, rastrskim terminalskim zaslonom in največjo resolucijo 1024 * 512. Partner bi zadostoval za osnovno spoznavanje delovanja grafike, vendar le do takrat, ko bi uporabnikove želje postale zahtevnejše. Žal Partner nima dovolj prostora za velike grafične pakete, ponavadi instalirane v HOST računalniku.

Takšni paketi so na primer GKS(Graphical Kernel System), CGI(Computer Graphics Interface), CGM(Computer Graphics Metafile), PHIGS(The Programmer's Hierarchical Interactive Graphics System). Za večino programskih paketov je značilno, da poskušajo biti čimbolj aparaturno neodvisni. Izjema je le krmilni del(DRIVER), ki skrbi za pravilno komunikacijo osrednjega neodvisnega dela s terminalom. Za vsak terminal ga je potrebno napisati znova. Da bi lahko napisali takšne programe za Partnerja, ga moramo uporabiti kot samostojen grafični terminal hkrati z HOST računalnikom, s katerim je povezan. Na ta način lahko kličemo grafične podprograme in funkcije v HOST računalniku in pri tem izkoriščamo vse grafične lastnosti mikroračunalnika Partner.

Za realizacijo te komunikacije je potreben protokol za čim hitrejši odtok in pritok informacij v mikroračunalnik.

2. PROTOKOL GRAFIČNE KOMUNIKACIJE

Protokol pomeni v splošnem dogovor o obnašanju komunikacije med računalniki oziroma terminali. Protokoli so odgovorni za generiranje in procesiranje podatkov. Najooli

znana mreža OSI/ISO(Open System Interconection / International Standard Organisation) na primer vsebuje sedem nivojev:

1. fizični nivo
2. nivo podatkovnih povezav
3. mrežni nivo
4. prenosni nivo
5. usklajevalni nivo
6. predstavitveni nivo
7. aplikacijski nivo

Za usklajevanje aplikacijskih nivojev uporabimo aplikacijski protokol; v tem okviru se giblje tudi protokol za grafično komunikacijo.

Osnovni in najbolj grobi princip protokola za grafično komunikacijo je v tem, da program v Partnerju prebere niz določenih znakov, poslanih s HOST računalnika, in pri tem pokliče ustrezni Partnerjev grafični ukaz. Kadar želimo risati na Partnerju brez pomoči HOST računalnika uporabljamo grafične ukaze s sistemske datoteke BIOLIB.PAS. V njej so podprogrami in funkcije za risanje na Partnerju. Pri tem je treba paziti le, da pišemo programe s katerimi kličemo te ukaze, v TURBO pascalu. Ukaze za komuniciranje z grafičnim procesorjem kličemo s pomočjo funkcije BIOS(29) - (Basic Input/Output System); od tod tudi ime datoteke.

S HOST računalnikom, v našem primeru z VAX-8800 in MikroVAX, želimo najti dostop do teh podprogramov in funkcij v datoteki BIOLIB.PAS v Partnerju.

Za doseg te realizacije, pošiljamo s HOST računalnika nek določen niz znakov.

Na Partnerju deluje program za zaznavanje znakov na vhodu/izhodu, pri spoznavnem nizu zaustavi nadaljno vnašanje in izvede ustrezno grafično funkcijo. Po potrebi pošlje tudi določen niz znakov nazaj k HOST računalniku.

3. REALIZACIJA PROGRAMA ZA PARTNER

Glavna naloga tega programa je torej, da bere niz znakov iz vhoda/izhoda v Partner, pri določenem spoznavnem nizu pokliče ustrezni grafični ukaz in po potrebi pošlje ustrezne parametre HOST računalniku.

Problem se pojavi pri branju in pisanju na vhod/izhod Partnerja, kjer je bilo potrebno tudi občasno zaustavljanje tega procesa. Ta problem je rešen s podprogrami in funkcijami, napisanimi v datoteki TEKUSR.INC. Ti nam omogočajo dokaj preprosto branje ali pisanje na vhod/izhod Partnerja. Razvili so jih strokovnjaki v Iskri - Delti v Ljubljani.

Pri iskanju spoznavnega niza grafičnih znakov ne smemo motiti ostalih nizov, ki označujejo ukaze, namenjene HOST računalniku. Poiskati je treba takšen niz znakov, ki se pri normalnih ukazih skorajda nikoli ne pojavi . Pri bolj zmogljivih grafičnih terminalih uporabljamo ubežne sekvence (escape sequence) za klicanje grafičnih funkcij. Partner tega ne pozna, zato smo se odločili za razpoznavni niz "%@". Znak "%" se zelo redko pojavi, vendar je še mnogo manjša verjetnost, da se bo za tem znakom pojavil znak "@". Pred vsakim nadaljnim ukazom stoji zopet znak "@" za kontrolo, če bi prišlo do kakšne motnje pri komunikaciji. S tem je dana možnost, da se ne zruši celoten sistem, ampak je napaka samo pri enem ukazu. Za znakom "@" sledi spoznavna koda ukaza.

S to kodo ugotovi program z imenom GRAFIKA, kateri ukaz bo moral poklicati iz datoteke BIOLIB.PAS. Tej kodi sledi niz znakov, ki predstavljajo parametre, potrebne za tekoči grafični ukaz. Niz parametrov se zaključi z znakom "@" za označitev začetka naslednjega ukaza. Za popolno zaključitev pošiljanje znakov grafike nam služi znak "Z".

Za izvajanje grafične komunikacije vtipkamo ukaz GRAFIKA, za zaključitev komunikacije uporabimo znak "<CTRL>\".

4. REALIZACIJA MODULA ZA VAX

Za pošiljanje in sprejemanje kod s HOST računalnika smo razvili modul z imenom GRAFPART.PAS. Modul nam omogoča klicanje ukazov z enakimi imeni, kakor pri BIOLIB.PAS v Partnerju tako, da pošlje ustrezne razpoznavne kode in njim pripadajoče parametre. Pred tem poskrbi za pretvorbo parametrov v znake.

Naravna števila se pretvorijo v niz številskih znakov in logične spremenljivke v posamezen znak (TRUE <=> "T"). Pri sprejemu pa se znaki pretvorijo v ustrezne vrednosti parametrov. V datoteki GRAFPART je spremenjeno le ime gtext v text, dodani pa so podprogrami InitWs, CloseWs in Update.

Za interaktiven vhod in izhod smo uporabili knjižnico SUNLET, točneje modul TTIO. Knjižnico so razvili na Institutu Jožef Stefan v Ljubljani.

Da bi bil komunikacijski program čim bolj učinkovit, se lahko držimo naslednjih pravil:

- poslane kode naj bodo čim krajše,
- za spoznavanje ukazov raje uporabimo črke namesto številk, s čimer prihranimo pomnilniški prostor,
- pri kodiranju oziroma pri dekodiranju uporabimo hitre algoritme,
- uporabimo knjižnico SUNLET-TTIO,

- za izpis znakov imamo tri možnosti:

a) uporaba ločila, ki se ne sme pojaviti v tekstu dvakrat zapored,
b) uporaba ločila, ki se v tekstu ne sme pojaviti,
c) uvedba parametra za število črk v nizu.

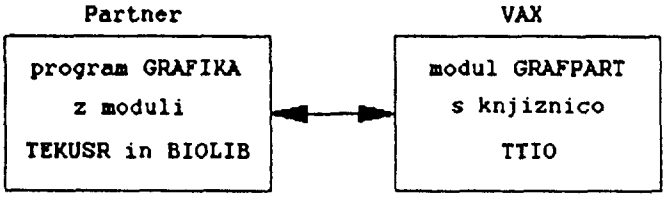
Za izpis teksta smo uporabili možnost pod c), ki je bila za to komunikacijo najbolj primerna. Naslednji primer nam prikaže pošiljanje kod s strani HOST računalnika (GRAFPART):

%@A1@C@H20 20 7GRAFIKA@B

%@	= začetna razpoznavna koda
A	= vklop grafike
1	= tip resolucije
@C	= brisanje zaslona
@H	= znak za tekst
20 20	= vrednosti za način pisanja
7	= število črk
GRAFIKA	= tekst
@B	= zaključek grafike

Ko dobi program GRAFIKA na Partnerju na primer niz "@A1", pokliče funkcijo ginit(1) z datoteke BIOLIB.PAS.

5. SHEMA PROGRAMOV:



6. SEZNAM UKAZOV IN NJIHOVO DELOVANJE

InitWS -	inicializacija delovne postaje in knjižnice SUNLET
CloseWS -	zaključitev dela z delovno postajo
ginit -	postavitev v grafični način in nastavitev resolucije
gexit -	zaključitev grafičnega načina
gclr -	brisanje grafičnega načina
gxy -	premik na pozicijo (x,y)
text -	pisanje grafičnega teksta
vbar -	nariše blok
circ -	nariše krog
disc -	nariše polni krog
ring -	nariše zapolnjen kolobar
draw -	nariše črto
vect -	nariše črto z relat. koordinatami
scroll -	premik slike po y-osi
getpix -	pove ali je točka na mestu, kjer se nahaja grafični kurzor prižgana
qfill -	polnjenje omejenih področij v vodoravni ali navpični smeri
getcursor -	vrne koordinate točke
Hardcopy -	kopira ekran na printer
Update -	ažuriranje slike

7. PRIMER

Naslednji primer nam prikaže uporabo podprogramov, napisanih na računalniškem sistemu VAX-8800.

Nariši poln krog s koordinatama x = 100 in y = 100 in s polmerom rad = 50 !

```
[INHERIT('grafpart')]
      { vstavitev ukazov za grafiko }
program primer;
begin
{ inicial. delovne postaje in SUNLET }
  InitWs;
{ inicializacija grafike 1024 * 512 }
  ginit(1);
{ risanje polnega kroga }
  disc(100,100,50);
{ zaključitev grafičnega načina }
  gexit;
{ zaključitev knjižnice SUNLET }
  CloseWs;
end.
```

Ko program zaključimo, ga prevedemo in nato povežemo z objektno datoteko GRAFPART.OBJ in knjižnico SUNLET.

8. ZAKLJUCEK

Ta komunikacija nam torej omogoča pisanje krmilnih programov za grafične pakete tako, da uporabljamo ukaze, ki jih pozna mikroračunalnik Partner. Seveda ne smemo pozabiti, da mora pri tem na Partnerju ves čas delovati komunikacijski program z imenom GRAFIKA.

Programi so napisani brez odvečnih znakov in poskušajo omogočiti čim hitrejšo komunikacijo med HOST računalnikom(VAX 8800, MikroVAX) in mikroračunalnikom Partner. Razlika v hitrosti delovanja grafičnega programa na samem Partnerju in hitrosti delovanja programa preko HOST računalnika je skorajda neopazna.

Knjižnico je uporabljena tudi za pisanje krmilnega programa GKS sistema za grafiko na Partnerju.

LITERATURA:

TURBO PASCAL,
Gams Matjaž:
Osnove dobrega programiranja,
VAX - 11 PASCAL:
Language Reference Manual,
VTO ERI:
Projekt-Mreže.

ZASNOVA RAHLO-SKLOPLJENEGA PORAZDELJENEGA SISTEMA ZA VODENJE INDUSTRIJSKIH PROCESOV

INFORMATICA 3/1988

UDK 681.3:62

Dragan Mrdaković, Primož Krajnik
Institut »Jožef Stefan«, Ljubljana

POVZETEK - V članku je predstavljena zasnova rahlo-sklopljenega porazdeljenega sistema z različnimi verzijami osebnih računalnikov za opravljanje posameznih nalog pri vodenju industrijskih procesov. Osnovne zahteve pri konstrukciji takšnega sistema so bile: - uporaba že obstoječe materialne in programske opreme, - primeren za vodenje v realnem času, enostavna instalacija, vzdrževanje in rekonfiguracija. Za povezavo postaj v sistem smo izbrali mrežo tipa ARCNET, kot najbolj primerno za industrijsko okolje.

BASICS OF LOOSELY COUPLED DISTRIBUTED SYSTEM FOR INDUSTRY PROCES CONTROL - This paper presents basics for loosely coupled distributed system for manufacturing automation based on various types of personal computers and input/output digital and analog extension boards. Basic needs in constructing such a system were: - use of existing hardware and software, simple implementation, maintenance, reconfiguration and flexibility. We choose ARCNET as an appropriate industry standard for factory floor communications between personal computers.

1. UVOD

Vse večja popularnost osebnih računalnikov pri raziskovalnem in inženirskem delu je vplivala na proizvajalce materialne opreme. Na svetovnem trgu je možno kupiti široko paleto zelo kvalitetnih vhodno/izhodnih modulov. Skupaj z ustreznim ohišjem in filterskimi hladilnimi napravami lahko uporabljamo osebni računalnik v industrijskem okolju za nadzor in vodenje procesov. Pomanjkljivosti PC vodila (vodilo osebnih računalnikov) so znane: - precej zaprta arhitektura v primerjavi s konkurenti (STD, VME, Multibus) in - omejena hitrost. Ne glede na to, izredno pestra že izdelana programska oprema za pisarniško okolje in nizka cena vpliva na izbiro osebnih računalnikov za vodenje industrijskih procesov. (npr. programi za tabelarično poslovanje lahko direktno sprejemajo podatke iz procesov za obdelavo, kot je LOTUS MEASURE). Glavna prednost osebnih računalnikov je njihova nezahtevnost programiranja, saj za konfiguriranje in aplikativno delo uporabniku ni potrebno podrobnejše znanje programiranja - včasih sploh nič. Uporabniki in programerji imajo danes na razpolago vrsto kvalitetnih programskih jezikov, z ustreznimi knjižnicami in pripomočki za testiranje svojih programov. Ponujajo se operacijski sistemi za vodenje v realnem času, kot so QNX, C-DOS, ... S pojavom novega več-opravilnega operacijskega sistema OS/2 firme MICROSOFT pričakujemo, da se bodo na široko odprla vrata za uporabo osebnih računalnikov pri vodenju industrijskih procesov. Zelo pomembna je tudi možnost povezovanja osebnih računalnikov v lokalno mrežo, kjer lahko računalniki med seboj izmenjujejo informacije za čim bolj racionalno in učinkovito vodenje procesov. Na IJS imamo že izkušnje s porazdeljenim sistemom DMS-860 glede povezave PC/XT/AT osebnih računalnikov v mrežo, primerno za vodenje procesov.

2. ZASNOVA RAHLO-SKLOPLJENEGA SISTEMA PRIMERNEGA ZA VODENJE INDUSTRIJSKIH PROCESOV

Pri zasnovi takšnega sistema za vodenje industrijskih procesov smo si zastavili naslednja izhodišča: /lit. 1,3,4,5/

1.) Uporabiti že izdelano standardno aparaturno opremo, ki je lahko dostopna in zamenljiva,
2.) Možnost uporabe že izdelane programske opreme, ki za izvedbo aplikacij ne zahteva preveč dodatnega dela - v čim večji meri izkoristiti obstoječo programsko opremo, ki dopušča enostavne in učinkovite spremembe za aplikacije,
3.) Povezati aparaturno opremo s standardno lokalno mrežo, ki je primerna za delo v realnem času v okolju z motnjami,
4.) Preprosta instalacija in vzdrževanje,
5.) Možnost enostavnega dodajanja novih postaj in funkcij v sistem,
6.) Vse to doseči s čim manjšimi potrebnimi vlaganji.

Naloge sistema so razdeljene v dve skupini:

- naloge, ki jih opravlja centralna mikroračunalniška postaja in
- naloge, ki jih opravljajo periferne postaje neposredno ob procesu.

Funkcije centralne mikroračunalniške postaje so:

- operaterju omogoča dostop do sistema in možnost izvajanja akcij v sistemu,
- prikaz stanja in izpis informacij o sistemu,
- alarmiranje, beleženje in arhiviranje alarmnih situacij in drugih važnejših podatkov o sistemu,
- komuniciranje z ostalimi mikroračunalniškimi postajami v sistemu - postaja ima lahko nekakšno monitorsko funkcijo nad sistemom.

Naloge perifernih mikroračunalniških postaj so:

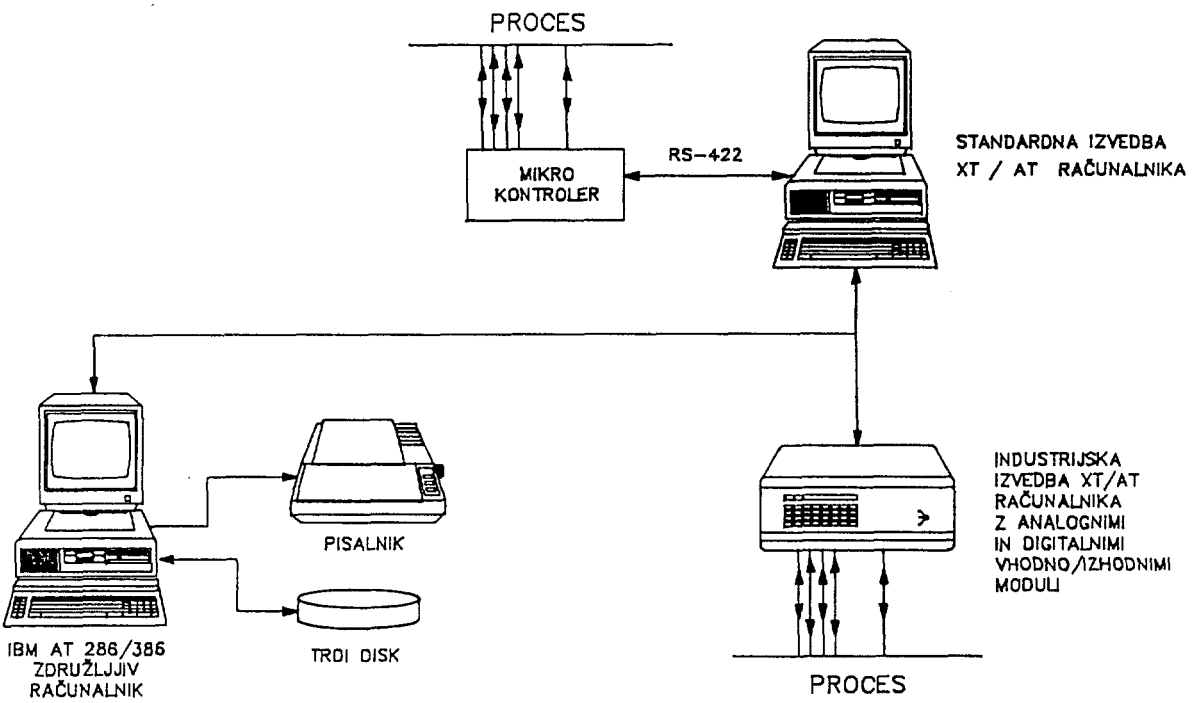
- zbiranje podatkov o proizvodnem procesu preko digitalnih in analognih vhodov (podatki iz senzorjev, števcev, merilnih pretvornikov, časovnikov, itd.),
- vključevanje posameznih izvršilnih členov (releji, ventili,...) z digitalnimi izhodi,
- start naprav z analognimi in sekvenčnimi izhodi,
- komunicira s centralnim mikroračunalniškim sistemom,
- obdeluje podatke o procesu in na osnovi algoritmov izvaja regulacijo in vodenje procesa.

Centralna mikroračunalniška postaja naj bo visoko zmogljiv IBM kompatibilni osebni računalnik tipa AT-386 ali AT-286 (na osnovi mikroprocesorja INTEL 80386 ali 80286). Na njem bo tekel več-opravilni operacijski sistem. Vgrajen bo pomnilnik - trdi disk večje kapacitete minimalno 40 Mbytov (običajno 80 Mbytov). Imel bi vgrajeno 5.25 ali 3.5 colsko disketno enoto ali vsaj možnost priključitve teh zunanjih enot za nalaganje programov in konfiguriranje. V določenih primerih bi lahko služila tudi za arhiviranje manjših datotek (arhiviranje alarmov, posegov v proces, oz. drugih važnejših podatkov o proizvodnji). Pri procesih z večjim številom podatkov oz. parametrov, bi lahko zaradi večje zanesljivosti in arhiviranja datotek večjega obsega dodali tračno "BACK-UP" enoto (streamer-tape) za občasno shranjevanje vsebine trdega diska. Za izpis podatkov o sistemu bo poleg pisalnik. Za prikaz aktivnosti in spremljanje določenih procesov in nazornejšega predstavljanja parametrov in rezultatov analiz bi poleg navadnega monitorja dodali še večji grafični barvni monitor (z 19-20 colsko diagonalo zaslona).

Z uporabo dovolj zmogljivega računalnika in uporabo več-opravilnega operacijskega sistema bi dosegli večjo uporabnost in izkoriščenost centralne postaje. Na njem bi lahko vodili podatke o materialnem poslovanju delovne organizacije - nabava surovin, stanje zalog, trend porabe surovin in zahteve po nabavi novih, podatki o proizvodnji polizdelkov, gotovih izdelkov, sprotno izračunavanje dejanske vrednosti izdelka in s tem donosnosti proizvodnje, ... S tem bi lažje optimirali tokove proizvajalnih sredstev. Uporabili bi ga lahko tudi za računovodsko-administrativne naloge. Če bi en sam centralni računalnik ne zmogel vsega tega, bi za pisarniška opravila lahko dodali še dodatne računalnike, lahko šibkejše (vnos podatkov in lokalni izračuni) in jih povezali z glavnim v mrežo, da bodo med seboj izmenjevali potrebne podatke. Vseeno je priporočljivo, da računalnike za vodenje procesov ne obremenjujemo s temi dodatnimi administrativnimi izračuni in v ta namen raje že takoj povežemo v mrežo računalnike za te namene, oz. tedaj, ko se pojavi potreba. Zaradi tega smo izbrali zelo fleksibilno mrežo za povezovanje računalnikov, da lahko spreminjamo konfiguracijo - dodajamo in odvzemamo računalniške postaje.

Za periferne postaje bomo glede na potrebe uporabili dve osnovni konfiguraciji IBM kompatibilnih PC/XT/AT računalnikov:

A.) Industrijska verzija PC/AT računalnika z vhodno/izhodnimi digitalnimi in analognimi moduli za neposredno povezavo s procesom. Takšen računalnik mora biti odporen proti agresivnemu industrijskemu okolju (prah, vibracije, električne motnje EMI/RFI), zato ne bi imel trdega ali gibkega diska. V primeru pomanjkanja pomnilnika bi dodali ploščo z delavnim pomnilnikom (RAM kartica z 2-16 Mbytov).



SLIKA 1. prikazuje zasnovo rahlo-sklopljenega sistema za vodenje industrijskih procesov, kot smo si ga zamislili na Odseku za energetiko in vodenje procesov na Institutu "Jožef Stefan" v Ljubljani.

B.) Standardna verzija IBM PC/XT/AT kompatibilnega računalnika, ki bo imel trdi disk manjše kapacitete - 30 ali 40 Mbytov in po potrebi še disketno enoto. Preko RS422 komunikacijskega vmesnika (do 10 Mb/s, diferencialni napetostni prenos) bi nanj priključili mikrokontroler (npr. na osnovi INTEL 8051 ali 8096 serije), ki bi krmilil svoj proces. Standard RS422 smo že uporabili na porazdeljenem sistemu DMS-860.

Izkušnje so pokazale, da je potrebno uporabiti čim bolj inteligenten kontroler za mrežni vmesnik, da bi razbremenili procesor in povečali hitrost prenosa na podatkovnem komunikacijskem vodilu mreže. /lit. 2/.

Za povezavo vseh teh postaj med seboj v enoten in celovit sistem pa potrebujemo lokalno mrežo. Tu smo postavljeni pred zahteve:

- mreža mora biti dovolj hitra - propustnost skupnega komunikacijskega kanala mora biti ustrezno velika,

- ker bo speljana v industrijskem okolju, mora biti prenos imun na elektromagnetne in radijske motnje (RFI/EMI). Izbrati moramo ustrezno kodiranje in fizični prenosni medij - kabel,

- delovanje v realnem času (zakasnitev med podano zahtevo in odgovorom druge postaje nanjo mora biti čim krajša in deterministično določljiva),

- omogočati mora enostavno priključevanje in rekonfiguriranje mreže (dodajanje novih postaj oz. opuščanje že obstoječih),

- mreža mora biti standardna (IEEE 802.X standard),

- sposobna mora biti povezovati postaje na večjih medsebojnih razdaljah.

Pri izboru mreže se lahko odločamo med CSMA/CD /lit. 6/ (Carrier Sense Multiple Access with Collision Detection - IEEE 802.3) protokolom, ki omogoča množičen istočasni dostop na skupno komunikacijsko vodilo z zaznavo trkov. Sem spadata najbolj znani mreži Ethernet (10 Mb/s, baseband, odseki v mreži so lahko dolgi do 500 m, če uporabljamo standardni koaksialni kabel) in StarLAN (1M b/s, baseband, odseki v mreži so lahko dolgi 250 m, oz. z uporabo vmesnih vozlišč 500 m ob uporabi prepletenih telefonskih paric) ali TOKEN-PASSING protokol - protokol s podajanjem žetona v token-bus (IEEE 802.4) ali token-ring (IEEE 802.5) izvedbi.

Tu sta najbolj znani ARCNET (token-bus 2,5 Mb/s, baseband, odseki z aktivnimi vozlišči do 600 m pri uporabi koaksialnega kabla) in IBM token-ring (4 Mb/s, uporablja oklopljene parice).

Po primerjavi in priporočilih MAP skupine (Manufacturing and Automation Protokol) - skupine, ki se ukvarja z razvojem in izdelavo standardov na področju lokalnih mrež za industrijsko okolje, smo se odločili za ARCNET, ki edina od teh deluje na osnovi token-bus (IEEE 802.4) načina dostopa do skupnega komunikacijskega vodila.

3. OPIS ARCNET LAN /lit. 1,5,7/

Arcnet je baseband (z enopasovnim prenosnim medijem) token-passing (s podajanjem žetona) lokalna mreža, ki se odlikuje z enostavnim povezovanjem, fleksibilno topologijo, zanesljivostjo in ustrezno propustnostjo. Izdelani sta dve vrsti vmesnikov, eni za medsebojno povezovanje IBM PC/XT/AT sistemov in drugi za povezovanje sistemov z VME vodilom.

ARCNET uporabniku zagotavlja naslednje funkcije in možnosti:

- ker temelji na token-passing protokolu je zakasnilni čas lahko deterministično določljiv v nasprotju s CSMA/CD, kjer obstaja le verjetnost dostopa določene postaje na vodilo.

- hitrost prenosa je 2,5 Mbitov/s, prenos je enopasovni - baseband, zaradi česar je vmesnik enostavnejši in cenejši,

- ARCNET ponuja različne možnosti povezovanja. Izbiramo lahko med topologijo zvezde, vodila ali kombinacije obeh. S tem se lahko približamo industrijskemu procesu in zmanjšamo stroške za kabliranje. Standardni prenosni medij je koaksialni kabel (baseband RG-62/U), lahko pa uporabimo tudi optična vlakna (če hočemo povečati razdaljo med postajami, doseči popolno imunost na motnje iz industrije in zagotoviti večjo zanesljivost prenosa).

- ARCNET je zelo prilagodljiva mreža. S svojo fleksibilno topologijo je možno preprosto izvesti dodajanje, odvzemanje ali prestavljanje mikroračunalniških postaj. Rekonfiguracija se izvrši avtomatsko kadarkoli odvzamemo ali dodamo postajo v mrežo, kar naredi ARCNET za eno najbolj zanesljivih mrež na tržišču v tem trenutku. Čas, potreben za rekonfiguracijo, se giblje med 21-61 ms, kar je odvisno od števila postaj in ID števila.

- Vzdrževanje in odpravljanje napak je enostavno. Aparaturno opremo, ki ne deluje, lahko enostavno odklopimo in, ko jo popravimo, vrnemo nazaj,

- z ARCNET lahko v osnovni konfiguraciji povežemo do 255 postaj z razponom do 6,4 km. Večje mreže dosežemo z uporabo mostičkov (bridge) - večje št. postaj ali uporabo optičnih vlaken za premostitev večjih razdalj.

- ARCNET je ena najlažjih mrež, kar se tiče instaliranja. Pravila povezovanja so zelo enostavna z le malo omejitvami.

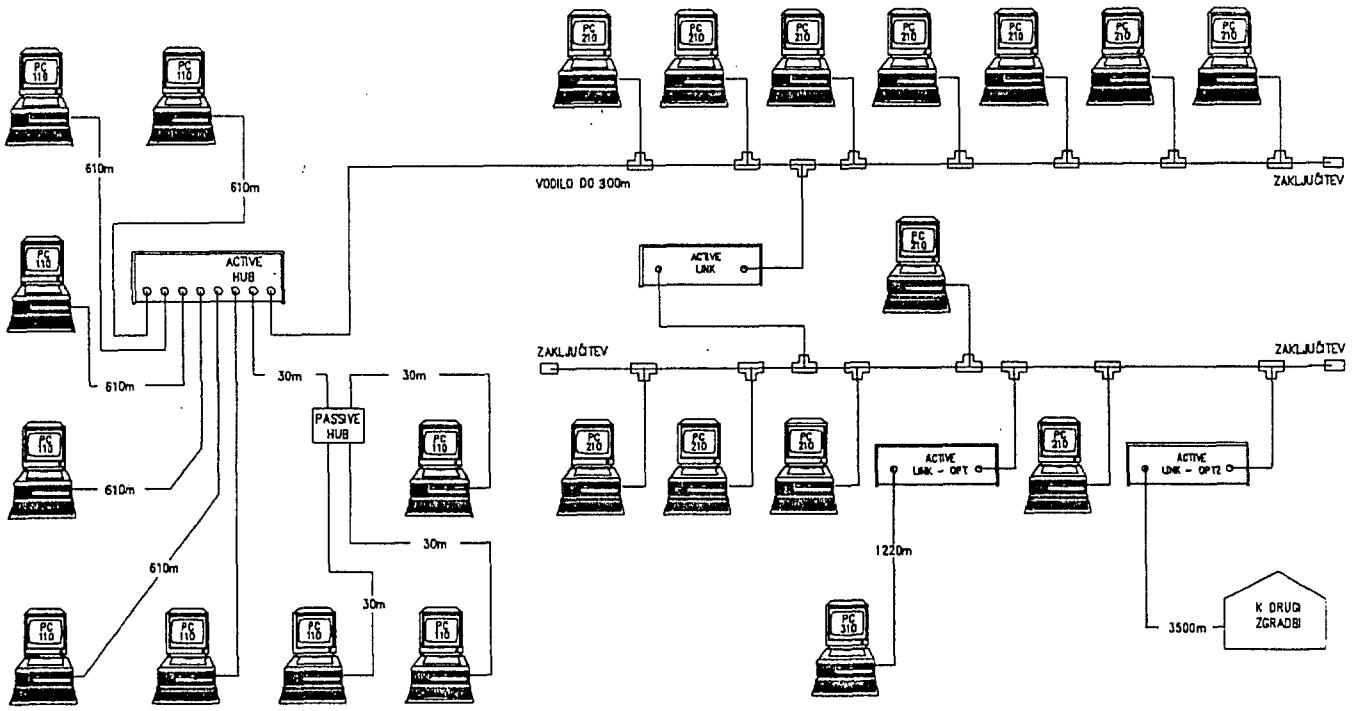
Pri mrežah na splošno je proces konfiguriranja mreže, dodajanja in odvzemanja delavnih postaj težka in zapletena naloga. Pri ARCNET mreži pa so vsi ti procesi implementirani hardversko z VLSI tehnologijo integriranih vezij. Ta ARCNET kontroler zato ne potrebuje posredovanja programske opreme, saj te funkcije izvaja avtomatsko. Tako ne obremenjuje procesorja v sami mikroračunalniški postaji. Procesor ima opravka samo še s sprejemanjem in oddajanjem sporočil. V starih izvedbah se je pojavljal problem "overhead-a", da so aktivnosti za mrežo vzele preveč časa procesorju , zaradi česar se je zmanjšala dejanska propustnost oz. hitrost prenosa na nivo 100.000 bitov/sek. Poleg tega je procesor imel manj časa za opravljanje svoje primarne vloge, npr. ni mogel skrbeti le za proces, ki ga je krmilil, zaradi česar se algoritem ni izvajal optimalno.

- ARCNET dejansko lahko uporablja skoraj vso operacijsko programsko opremo za mreže.

4. POVEZOVANJE OSEBNIH RAČUNALNIKOV V A R C N E T MREŽO

ARCNET omogoča topologijo zvezde, vodila ali kombinacijo obeh. /lit. 7/ Zvezdasta struktura se priporoča za prostorsko medseboj bolj oddaljene postaje. V osebni računalnik se vgradi ARCNET vmesnik tipa 110. Postaje povezujemo med seboj preko centralnih vozlišč. Obstajata dva tipa vozlišč za povezovanje postaj: - 8-vhodno aktivno in - 4-vhodno pasivno vozlišče. Razdalja med aktivnim vozliščem in postajo lahko z uporabo koaksialnega kabla doseže 620 m, pri pasivnem vozlišču pa le 30 m. Za fizični medij pri povezovanju lahko uporabljamo v isti mreži hkrati koaksialne kable in optična vlakna. Priključevanje je izvedeno z navadnimi BNC konektorji.

Pri topologiji vodila se uporablja koaksialni kabel z BNC T konektorji. Topologija vodila je zelo ekonomična pri povezovanju postaj, ki se nahajajo blizu skupaj. Tu ne potrebujemo vmesnih aktivnih vozlišč in kabli so krajši. Posamezen odsek je dolg 300 m in nanj lahko priključimo do 8 postaj v katere vgradimo ARCNET 210 vmesnik tipa 210. Te vmesnike za večje



SLIKA 2. prikazuje primer ARCNET mreže z vpisanimi tipi vmesnikov instaliranih v osebnih računalnikih, vozlišč, vmesnikov za povezovanje in možne največje razdalje med sestavnimi deli mreže.

konfiguracije povezujemo z aktivnimi dvo-vhodnimi vmesniki, ki jih je več vrst, za povezovanje odsekov s koaksialnim kablom ali optičnimi vlakni (do 3,6 km odsek, ki lahko služi povezovanju različnih stavb ali bližnjih dislociranih obratov).

5. ZAKLJUČEK

Pojava široke palete vhodno/izhodnih modulov in že obstoječa programska oprema opravičujejo uporabo osebnih računalnikov v industrijskem okolju. Z njimi smo dosegli osnovne cilje pri zasnovi rahlo-sklopljenega sistema primernega za vodenje industrijskih procesov kot so: - uporaba že izdelane aparaturne in programske opreme primerne za delo v realnem času, - enostavna instalacija in vzdrževanje in podobno.

Hrbtenica takšnega sistema je komunikacijsko vodilo. Odločili smo se za mrežo tipa ARCNET (več kot 500 000 vgrajenih vmesnikov v svetu, oz. 25 %-ni delež na tržišču mrež). Omogoča dokaj hiter prenos informacij, deterministični dostop do vodila, različne topološke strukture, uporabo optičnih vlaken in koaksialnih kablov v isti mreži, enostavno rekonfiguriranje. Izkušnje na Institutu "Jožef Stefan" so pokazale, da mora biti vmesnik za mrežo čim bolj samostojen - z inteligentnim kontrolerjem, da razbremeni glavni procesor. Za mrežo tipa ARCNET je razvit niz komponent, ki opravljajo vse naloge vazane na vzdrževanje žetona, konfiguracijo in rekonfiguracijo sistema.

Osnovna karakteristika predloženega rahlo-sklopljenega sistema je njegova fleksibilnost. Sistem omogoča prilagajanje konkretnim potrebam, enostavno nadgrajevanje in priključitev novih izdelkov. Pričakujemo, da bo zadoščal tudi za vodenje zelo zahtevnih industrijskih procesov.

LITERATURA:

/1./ Colin Bartram: An ARCNET System Links Multivendor Robots, Control Engineering, 2.October 1987, stran 38-39;

/2./ D. Mrdaković, M.Tomšič, M. Vidmar: Osnovne karakteristike distribuirane višemikroračunarske mreže DMS-860, Tehnika, 10, 1985;

/3./ Dick Lefkon: A LAN Primer, Byte, July 1987, stran 147-154;

/4./ Donald A. Dytewski: Planning Remains the Necessary Ingredient to make MAP work, Control Engineering, 2.October 1987, stran 19-20;

/5./ George Thomas: Arcnet Simplifies Factory Floor Communications, Control Engineering, 2.October 1987, stran 40-41;

/6./ INTEL: Local Area Networking (LAN) Component User's manual, September 1985;

/7./ Standard Microsystems Corporation: ARCNET Overview, 1987.

Darko Kodrič
Iskra Elementi TOZD Hipot
Šentjernej

UDK 681.326

Članek predstavlja Komuniacijski vmesnik KV8, namenjen povezavi različnih računalniških sistemov po RS232 standardu. Komunikacijski vmesnik KV8 omogoča priključitev in povezavo do 8 sistemov, ki lahko komunicirajo med sabo preko tega vmesnika.
V članku je podan princip delovanja komunikacijskega vmesnika KV8.

Communication Interface KV8

This paper presets communication interface KV8, used for connection of maximum 8 different systems through RS232 interface. All systems, connected through KV8, can communicate to each other, but maximum 4 connections can be made in the same time.
Paper presents principles of operation of this communication interface.

1. Uvod

Komunikaciski vmesnik KV8 je bil razvit za povezavo različnih računalnisko krmiljenih merilnih sistemov s poslovnim sistemom, za prenos podatkov o kakovosti proizvodnje in merilnih rezultatov med posameznimi merilnimi sistemi in za prenos podatkov za poslovno informatiko. Komunikacijski vmesnik KV8 je zgrajen modularno, tako aparaturna, kot programska oprema, da je enostavna širitev vmesnika, če je to potrebno. Komunikacija lahko poteka med posameznimi kanali, pri tem je možno istočasno prenašati po 4 poteh, če se vzpostavijo komunikacije med različnimi kanali.

Poleg osnovnih 8 kanalov ima komunikacijski vmesnik KV8, še dodaten serijski kanal, 20 mA tokovna zanka, za priključitev terminala, ki omogoča v nadzornem načinu dela spremeniti parametre serijskih vmesnikov za posamezne kanale. Preko prvega kanala, ki je vezan na poslovni sistem, je možno vzpostaviti tudi direktno komunikacijo, za normalno delo terminala na prvem kanalu.

2. Materialna oprema Komunikacijskega vmesnika KV8

Materialno opremo Komunikacijskega vmesnika KV8 tvorijo moduli CPE, 16Kbyte RAM pomnilnik, in 4 moduli z dvema serijskima vmesnikoma.(slika 1) Širitev materialne opreme je možna z dodajanjem modulov z dvema nadalnjima serijskima kanaloma.

Na komunikacijskem vmesniku je možno poljubno nastaviti na vsakem kanalu RS232 serijsko komunikacijo, ali pa 20mA tokovno zanko (tty).

2.1. CPE (centralna procesna enota)

CPE vsebuje poleg 8 bitnega mikroprocesorja še 1 K byte RAM pomnilnika, v katerem so locirani sklad, vsi statusi vmesnikov, statusi povezav, naslovi pomnilniskih vmesnikov vseh kanalov in ustrezni kazalci, ki omogočajo komunikacijo med posameznimi kanali komunikacijskega vmesnika. 4Kb EPROM pomnilnika vsebuje vse potrebne programe za delovanje komunikacijskega vmesnika KV8. Na CPE enoti se nahaja še serijski komunikacijski adpter za priključitev terminala (20mA tokovna zanka), (slika 2), za nastavitev parametrov.

2.2. 16 Kbyte RAM pomnilnik

Pomnilnik je razdeljen v bloke 1 Kbyte za sprejem in 1 Kbyte za oddajo na kanal. Zato morajo biti paketi podatkov za prenos manjši od 1 Kbyte znakov. Ločen pomnilnik sprejemnika in oddajnika pa omogoča sprejem podatkov na posameznem kanalu se preden je linija z oddajnikom vzpostavljena.

2.3. Serijski vmesniki

Na vsakem modulu sta dva serijska vmesnika, ki sta programsko nastavljiva. To sta vezji SN2651, ki jima je možno programsko nastaviti hitrost prenosa, dolžino besede, pariteto, število stop bitov in način dela. Za vsak modul je možno izbrati poljubni 8 bitni naslov, ki definira številko kanala za programski dostop do vmesnika. Nastavitev komunikacije vsakega kanala na RS232 ali na 20 mA tokovno zanko je možno z nastavitvijo kratkospojnikov na modulih z vmesniki. Ob inicializaciji so vmesniki nastavljeni na 7 bitno dolžino besede, hitrost prenosa 1200 baud, 1 stop bit, pariteta prepovedana.

3.Programska oprema

Programski paket sestavljajo številne rutine, ki omogočajo inicializacijo komunikacijskega vmesnika, čitanje znaka posameznega kanala v sprejemni buffer tega kanala, prenos znaka iz oddajnega bufferja v serijski vmesnik, prenos znaka iz sprejemnega vmesnika kanala I v oddajni vmesnik kanala J, če je vzpostavljena povezava med kanalom I in J. Rutine testirajo zasedenost oddajnika pri vzpostavitvi zveze sprejemnika in oddajnika, vzpostavitev zveze, prekinitev zveze ob koncu prenosa podatkov med kanaloma I in J, kjer velja I <> J, ter I , J = (0, 1, ... , 7).
Za vse rutine je številka kanala vhodni podatek za določitev odmika v tabeli parametrov kanala in tabeli kazalcev na vhodne in izhodne pomnilniške vmesnike ter flage povezav.

Opis posameznih rutin:

3.1. Inicializacija

Inicializira vse vmesnike na hitrost prenosa 1200 baud, dolžino besede 7 bit, 1 stop bit, ni paritete, vse povezave so dovoljene, nobena povezava ni vzpostavljena, podatkovni vmesniki so prazni, poslan je 'ctrl Q' ukaz v vse serijske vmesnike - dovolitev prenosa. Čita status načina delovanja in če je v nadzornem načinu delovanja omogoča delo prikljucku na CPE modulu.

3.2. Sprejem

Rutina čita status serijskega vmesnika izbranega kanala in če ta vsebuje znak, ga prenese v sprejemni vmesnik tega kanala. Istočasno testira znak in v primeru, da je 'ctrl S' ali 'ctrl Q' vrne znak med parametre za prenos podatkov preko tega kanala, kar definira flag za dovolitev oddajanja na kanalu.

Prav tako testira, če je sprejet znak STX, za določitev povezave sprejemnika z oddajnikom - po sprejemu tega znaka je možno vzpostaviti povezavo sprejemnika in oddajnika prenosa.

3.3. Oddaja

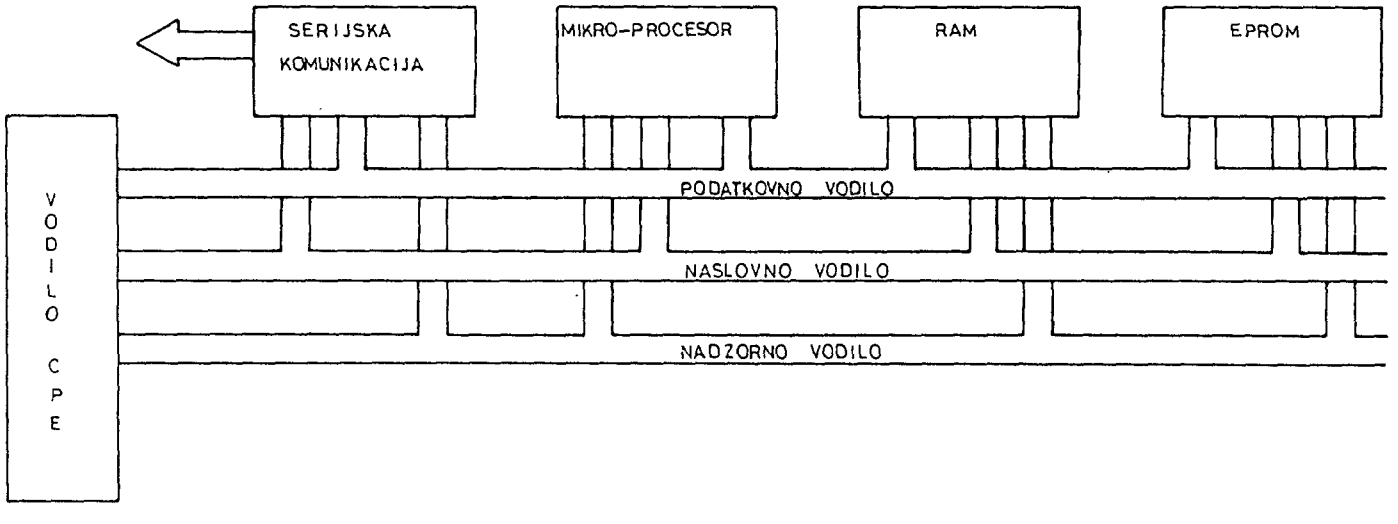
Ta rutina prenaša znake iz oddajnega bufferja k serijskemu vmesniku kanala. Ta prenos poteka toliko časa, dokler ne zazna v pomnilniku zapisan znak EOT - konec prenosa. Ob prenosu vsakega znaka ugotavlja, če status dovoljuje nadaljevanje prenosa.

3.4.Kopiranje

Ko je določena povezava že vzpostavljena in so določeni vsi kazalci, ki bodo omogočili prenos podatkov iz sprejemnega vmesnika kanala I v oddajni vmesnik kanala J, kopirna rutina prenaša podatke v ustezni vmesnik. Naslov ponora podatkov je določen prehodno z rutino, ki ob znaku STX ugotovi, če je možno vzpostaviti zvezo in jo, ko je možno, tudi vzpostavi. Ob koncu kopiranja iz kanala I v kanal J , ob znaku EOT, se zveza razdre, ko pa so vsi podatki preneseni, pa bo oddajni del kanala J zopet na razpolago za prenose, kanal I pa je prost za delo že ob koncu kopiranja podatov.

4.Prenos podatkov

Prenos podatkov iz kanala I v kanal J je mogoč takrat, ko je v kanalu I možen sprejem znakov, kar pomeni, da je kanal prost in je bil na kanal I poslan znak 'ctrl Q' . S tem lahko prične sprejem znakov v sprejemni buffer kanala I. Kanal J pa lahko prične sprejemati znake od kanala I v trenutku, ko je končal prenos znakov, ki jih je imel v oddajnem bufferju kanala



Slika1 : BLOKOVNA SHEMA CPE

J. V tem trenutku se najprej vzpostavi zveza med tema dvema kanaloma, istočasno so lahko 4 zveze, ki omogočajo prenose podatkov v KV8. Po vzpostavitvi zveze je postavljen flag, ki dovoljuje kopiranje, iz kanala I v kanal J, zveza pa je zapisana v parametrih obeh kanalov, in sicer v kanalu I, je vpisana vrednost J, v parametrih kanala J pa vrednost za kanal I. Vsak prost kanal ima v parametrih zveze kanala vrednost FFH.
Do vzpostavitve zveze je lahko prišlo po sprejemu znaka STX v sprejemnem vmesniku kanala I, sprostitev zveze pa je mogoča po prenesenem znaku EOT pri kopirni rutini. Nove zveze pa kanal J ne bo vzpostavil, dokler ne bo prenesen kompleten blok podatkov.

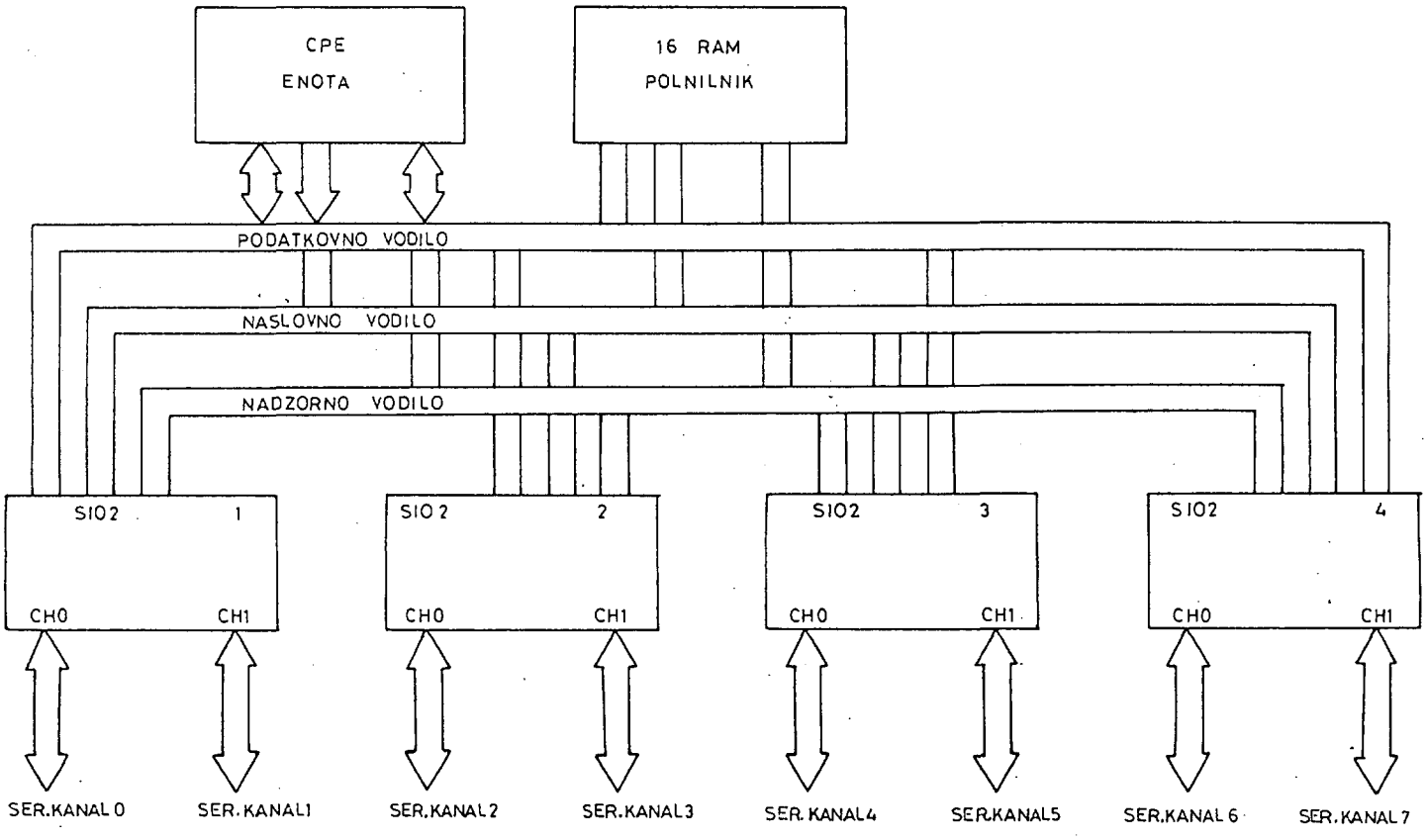
5. Sklep

Prenos podatkov v Komunikacijskem vmesniku KV8 je izveden v pooling načinu delovanja celotnega mikroračunalniskega sistema, možno pa bi bilo popraviti sistem na interruptni način delovanja, vendar je za izbrano aplikacijo, kjer je relativno majhno stevilo prenesenih podatkov, ki pa se tudi malokrat prenašajo, ustrezna resitev popolnoma zadovoljiva. Vsi vmesniki so večji, kot je maksimalno stevilo prenesenih podatkov v eni transakciji, zato je bil izbran tak način dela Komunikacijskega vmesnika KV8.

NASLOV SPREJ.	NASLOV ODDAJ.	PODATKI NADZOR	STX	PODAT. BLOK ZA PRENOS	ETX	EOT

Slika 3: Struktura podatkov za prenos v KV8

Ta vmesnik bo omogočal v aplikaciji prenose podatkov med merilnimi sistemi v proizvodnji Hibridnih vezij, omogočal bo povezavo le teh z laserji za aktivno justiranje uporov na hibridnih vezjih ter povezavo merilnih mest z poslovnim računalniškim sistenom, za prenos podatkov o opravljenih meritvah kvalitete izdelkov.



Slika2: BLOKOVNA SHEMA KVM 8

FORUM INFORMATIONIS

FORUM INFORMATIONIS V ČASOPISU INFORMATICA

Forum Informationis (FI) je nova, stalna rubrika časopisa Informatica, ki bo prinašala polemične, kritične in druge aktualne prispevke s področja računalništva in informatike. Tudi Slovensko društvo Informatika je na seji svojega Izvršnega odbora, dne 8. 4. 1988, ustanovilo interesno skupino z imenom *Forum Informationis*, ki naj pospešuje in razvija javno razpravo o bistvenih problemih domačega in tujega informacijsko-razvojnega komleksa, ki ga sestavljajo npr. problematika industrije, raziskovanja, izobraževanja, uporabe informacijskih sredstev, publicistike, zakonodaje, integracijskih procesov, drobnega gospodarstva, vodenja, zaščite, ekonomike, poslovanja, politike, mednarodnega povezovanja itd. Pri tem gre za strokovne, intelektualne, kulturne, razvojne in druge vidike v kontekstu socialne relevantnosti računalništva in informatike. Časopis Informatica se tako ustvarjalno vključuje v krizno in razvojno polemiko sodelavcev in bravcev časopisa Informatica.

KRATKO POROČILO S PRVEGA ZASEDANJA FORUM INFORMATIONIS

Prvi sestanek FI je bil za okroglo mizo, dne 5. maja 1988, v sejni dvorani SOZD Iskra, v Ljubljani, Trg revolucije 3/XIV. Na ta začetni sestanek je bilo vabljenih 42 strokovnjakov, novinarjev, zasebnikov itd. s področja računalništva in informatike. Udeležba je bila polovična s prisotnotjo 20, in sicer: Božidar Blatnik (Intertrade), Petar Brajak (I Delta), Vanja Bufon (I Delta), Rado Faleskini (SOZD Iskra), Bogo Horvat (TF Maribor), Marko Jagodič (I Telematika), Andrej Jerman-Blažič (I Telematika), Srečko Klančar (I Avtomatika), Mitja Kosirnik (študent FE), Milan Mekinda (I Mikroelektronika), Peter Mirkovič (Teleks), Alenka Mišič (GZS), Saša Prešern (I Delta in IJS), Mija Repovž (Delo), Ivan Rozman (TF Maribor), Uroš Stanič (IJS), Bruno Stiglic (I Electronics, Santa Clara, Ca), Marko Valič (I CEO), Jernej Virant (FE) in Anton Zeleznikar (I Delta). Večina povabljenih neprisotnih se je opravičila.

Prisotne sta pozdravila predsednik Slovenskega društva Informatika Milan Mekinda in član KPO SOZD Iskra za raziskave in razvoj Rado Faleskini. Podpisani je poskušal navesti razloge, ki so pripeljali do ustanovitve FI. Konec prejšnjega leta so se inženirji Iskre Delte resno začeli soočati s problemom finančne nezadostnosti Iskre Delte pa tudi s trendi demotivacije zlasti v raziskovalno-razvojnem sektorju. Z izjemo projekta Parsys, ki je reševal svoje preživljanje s potencirano samoorganizacijo in s stiki s svetom, so ostali veliki razvojni projekti stagnirali, se osipali in drseli v svoje prenehanje. Hkrati je postajalo očitno nesorazmerje med "napredovanjem" znanosti in "nazadovanjem" industrije, med znanstveno samonamembnostjo in njeno finančno potešenostjo na eni in kriznostjo industrije na drugi strani. Znanost je dosegla finančno neodvisnost od industrijske neuspešnosti predvsem z delovanjem svojih lobijev.

Vrstni red razreševanja krizne problematike v okviru FI za letošnje leto bi lahko bil naslednji: industrija, raziskovanje in izobraževanje. Če računalniška industrija in z njo povezana infrastruktura ne bosta preživeli, se prav lahko začne postavljati vprašanje o smiselnosti nadaljevanja raziskovalnih in izobraževalnih naporov kompleksa računalništva in informatike. (Verjetno bi Slovenci s tem lahko vstopili v zgodovino človeštva kot ena prvih civilizacij, ki se je zavestno odpovedala lastnim možnostim oblikovanja informacijske družbe.)

Namen prvega FI je bil informativen. Sprejeta pa so bila priporočila, da se takoj oblikujejo interesne skupine za pripravo koreferatov, ki bodo obravnavali področje industrije, izobraževanja in raziskovanja. Posameznim skupinam naj bi zaenkrat predsedovali podpisani, prof. dr. Jernej Virant in doc. dr. Saša Prešern. Za področje industrije naj bi bil sklican naslednji FI konec junija. Predloženo je bilo (Mija Repovž), da se k delu industrijske skupine pritegnejo tudi mlajši, "nedogmatični" ekonomisti. V korpus te skupine so bili predlagani Bufon, Faleskini, Gerkeš, Jerman-Blažič, Kajzer, Leskovar, Rupnik, Sočan, Sprah, Voh, Vovk itd.

Dnevno časopisje je poročalo o prvem FI (Delo dvakrat in Teleks). Teleks je ocenil FI kot provokacijo, čeprav se podpisani bolj nagiba k oceni reševanja, kar se rešiti še more. Predvsem pa ostaja funkcija FI prej ko slej oblikovanje nove intelektualne (dejavne) zavesti na področju RiI.

A. P. Zeleznikar

Odprto pismo uredniškem odboru časopisa Informatica

Spoštovani:

Že vseh dvanajst let občasno spremljamo razvoj časopisa Informatice. Po svojih močeh si namreč tudi sami prizadevamo za razvoj računalništva in vsaj tistega dela informatike, ki je vezan na računalništvo, na katerega se vsaj malo spoznamo.

Časopis Informatica je igral pomembno vlogo pri razvoju računalništva pri nas. Do pojava Mojega mikra (in Bita) je bil edina slovenska revija, posvečena računalništvu. Seveda je takih pomembnih dogodkov v razvoju računalništva pri nas še veliko. Spomnimo se le kongresa IFIP, ki je leta 1970 pripeljal v Ljubljano celotno svetovno računalništvo. Potem je tu še ustanovitev RRC, pa začetek visokošolskega študija računalništva, ki ga izvajata FE in VTOZD matematika in mehanika, pa začetki slovenske računalniške industrije in še bi lahko naštevali.

Bojimo pa se, da vloge časopisa Informatice v današnji situaciji ne razumemo več. Zato bi prosili predvsem glavnega in odgovornega urednika, mogoče pa še koga drugega izmed urednikov ali pa članov zaenkrat še anonimnega **Forum informationis** za odgovore na naslednja vprašanja.

1. Kakšen je koncept in kakšna je perspektiva časopisa Informatice? Komu je namenjen? Če gre za strokoven časopis, namenjen jugoslovan-

skim strokovnjakom, čemu so potem v njem prispevki v angleščini, nekateri celo s povzetki v japonščini(!)? Ali pa gre nemara za znanstven časopis? Kakšna je tedaj znanstvena raven tega časopisa? Ker niti med uredniki niti med avtorji ni zaslediti videti uglednih tujih raziskovalcev, pa tudi nikjer v tujini še nismo videli, da bi se kdo skliceval na rezultate, objavljene v Informatici, bi lahko sklepali, da gre za tretjerazredni znanstveni časopis, ki v znanstvenem svetu nima nikakršnega odmeva. Veseli bomo, če nas prepričate, da so naši sklepi napačni.

2. Kakšni recenziji so podvrženi prispevki za časopis Informatico? Kolikšen odstotek prispevkov je zavrnjen? Kakšna je bila vloga tehničnega odbora in čemu je bil z desetim letnikom le-ta ukinjen? Ali so prispevki lektorirani? Kako časopis skrbi za slovensko računalniško terminologijo? Kdo skrbi za angleščino?

Naslednje vprašanje pa je glavni razlog, da smo se odločili za pisanje tega pisma.

Oglejmo si najprej nekaj citatov iz zadnjih dveh številk:

a) "Umetna inteligenca prihaja na sam rob racionalizma, ko govori o inteligenci, ki pa ni več racionalistična kategorija. Razumevanje inteligence je povezano z novim pojmovanjem informacije. Stroji, ki bodo imeli vgrajeno inteligenco, ne bodo več zgolj-racionalistični." Informatica 1/88, str. 5.

b) "We hope to (be) able to receive your technical reports and to exchange information in the future." Informatica 1/88, str. 79.

c) "We look forward to fruitful joint research" Informatica 1/88, str. 81.

d) "Have I really done nothing in the last two years" Informatica 1/88, str, 91.

e) "The goal of this preliminary discourse is, of course, to lay out the way which leads to something termed informational logic. In the framework of such new logic, it will become possible to develop essentially changed concepts of set, relation, function, category, etc. as was the case in today's and yesterday's mathematics.", Informatica 2/88, str. 31.

f) "Zaključek tega razmišljanja je torej tak, da tako človek kakor stroj nista determinirana in imata ustvarjalne sposobnosti", Informatica 2/88, str. 116.

Omenjene citate smo izbrali zato, da bi ugotovili, da stil prispevkov v Informatici močno odstopa od stila običajnih uglednih mednarodnih znanstvenih in strokvnih revij. Podobnih citatov bi lahko nabrali predvsem v zadnjih letnikih še precej več.

V citatu a) gre očitno za filozofsko spekulacijo. Ker pri nas obstajajo takim problemom primerne revije (npr. Anthropos), bi po našem mnenju veljalo take prispevke objavljati tam.

V citatih b) in c) je opaziti, da gre za obljube, torej za nekaj, kar naj bi se zgodilo v prihodnosti. Mislimo pa, da za take obljube ni mesta v resni reviji. Pričakujemo, da bomo o konkretnih uspehih projekta Parsys kmalu brali v Informatici in v drugih, mednarodnih revijah (v nasprotnem primeru pa gre za razmetavanje z denarjem - vsaj kar se prostora v Informatici tiče).

Citat d) kaže na polemiko. Mislimo, da polemika brez strokovnih argumentov ne sodi na strani Informatice.

Citata e) in f) pa imata skupno predvsem to, da dišita po lažni znanosti. Prispevka, iz katerih sta vzeta citata, po našem mnenju namreč nista znanstvena. Morda sodita v filozofijo. V tem primeru bi ju ponudili npr. spet Anthroposu. Morda sodita tudi v znanstveno fantastiko, vendar o tem težko sodimo, ker se na znanstveno fantastiko ne spoznamo. Na vsak način, nikakor ne sodita v Informatico.

V Informatici dajeta vtis, kakor da gre za pravo znanost. Dopuščamo seveda možnost, da se motimo. Kaj pa če je res mogoče dokazati, da ima stroj ustvarjalne sposobnosti(!)? Mogoče pa gre res za temelje čisto nove logike, ki bo pospravila z obstoječo logiko in matematiko. V obeh primerih se čudimo skromnosti avtorjev, da sta za tako epohalna odkritja izbrala revijo s tako majhno odmevnostjo.

Menimo, da avtorja iskreno verjameta v to, kar pišeta. Zato ju rotimo, da pošljeta prispevka v angleščini v ponovno objavo. Če se ne moreta odločiti, katero od uglednih revij izbrati priporočamo, da prispevke pošljeta prijateljem: M. Kalosu, D. Sieworeku, Z. Segallu ali T. Winogradu (glej njihove slike na straneh Informatice), ki bodo že vedeli, kje je pravo mesto za take rezultate. Če so omenjeni rezultati resnični, bi avtorja zaslužila vsa najvišja odličja, vključno z Nobelovo nagrado.

In zdaj še vprašanje.

3. Zakaj Informatica objavlja prispevke, ki se prikazujejo kot znanost, pa to niso?

Za konec še pojasnilo. Ko govorimo o primernosti in neprimernosti prispevkov za Informatico, imamo v mislih uredniško politiko in strokovno recenzijo, nikakor pa ne cenzure. Mislimo, da bi družba morala poskrbeti za objavo še tako spornih prispevkov - vendar nismo prepričani, da ravno na straneh Informatice, če le-ta ne redefinira svojega koncepta. (npr. časopis za računalniško filozofijo in znanstveno fantastiko). Alternativa bi bila, da bi vpeljali rubrike za računalniško filozofijo, računalniško znanstveno fantastiko in računalinški humor, v katerih bi lahko brez zavajanja bralcev objavljali ustrezne prispevke seveda ob strokovni recenziji in ob obveznem lektoriranju vseh prispevkov.

Z veseljem bi vam poslali tole pismo z elektronsko pošto, pa nimamo vašega naslova. Mogoče bi lahko v eni prihodnjih številk pojasnili, kako vam lahko elektronsko pošiljamo prispevke.

V upanju, da pismo razumete v smislu natančnejšega določevanja uredniškega koncepta časopisa Informatice, vas lepo pozdravljamo.

Tomaž Pisanski,
Vladimir Batagelj, Marko Petkovšek, Marjan Špegel, Boštjan Vilfan, Iztok Tvrdy

O ODGOVARJANJU NA ODPRTO PISMO

Odprto pismo, ki ga objavljamo, načenja vrsto vprašanj, ki ne zadevajo le problematike časopisa Informatica, temveč se razprostirajo preko domene aktivnosti, ki jih časopis le reflektira. Vsako poenostavljeno odgovarjanje, ki bi

bilo brez nadaljnega mogoče, bi hkrati podpiralo tudi sprenevedanje in blasfemijo podpisnikov pisma in tako prav gotovo ne bi odkrivalo izčrpnejših in za prakso strokovnih in drugih dejavnosti informatike (in računalništva) uporabljivih odgovorov. Predvsem bo na vsa vprašanja piscev pisma odgovorjeno, vendar se to zaradi omejenih možnosti hkratnega odgovarjanja ne bo zgodilo v enem samem izčrpnem paketu.

Nekateri očitki so pač takšni, da jih ne bo mogoče dokazovati le z navedbami iz domačih logov pa tudi ne tako poenostavljeno, kot to upajo pisci pisma. Pri tem bo potrebno poseči v navedbe (in prakso) podobnih (celo avantgardnih) strokovnih časopisov po svetu (v mislih imam npr. časopise kot so Artificial Intelligence, New Generation Computing, AI Communications, IEEE Computer, IEEE Transactions on System, Man, and Cybernetics itd., ki so na svojih področjih strokovne in tudi znanstvene relevance - beri računalniške in informacijskevodilni na svetu). To pa bo zahtevalo nekoliko več dela, ki ne bo brezplodno in bo prav gotovo koristilo domačim (T. Pisanski bi bržkone hitel poudariti "tretjerazrednim") razmišljanjem.

Seveda pričakujem, da bodo v kritični dialog, ki se s tem začenja, posegli tudi bravci. Končno se začenja tudi to, kar bi lahko imenovali "normalni polemični in kritični dialog na področju informatike in računalništva". Ta dialog je seveda zaželjen in časopis Informatica ga mora podpirati v razumnih mejah, dokler se vpraševanje odprtega pisma ne bo izčrpalo. Seveda prosim pisce pisma za določeno potrpežljivost, da počakajo na sadove njihovega vpraševanja.

Urednik

NEZNANSTVENO KRATEK ODGOVOR UREDNIKA

Znanost lahko odgovarja le z dejstvi, ta pa so v primeru tudi na urednika naslovljenega pisma kratko tale (po vrstnem redu vpraševanja):

1. Koncept časopisa Informatica je stroka, znanost, raziskovanje, obveščanje, izobraževanje, industrija, proizvodnja, razvoj, zanimivosti, novosti iz računalništva in informatike itd. Perspektiva časopisa je lahko samo njegova zadostna aktualnost na področju opisanega koncepta. Časopis je namenjen inženirjem, raziskovalcem, razvijalcem, študentom, podiplomcem, tudi vodilnim učiteljem itd. na področju koncepta. Da pa v tako opisanem okolju koncepta ni filozofskega, se verjetno ni mogoče strinjati (našemu doktoratu znanosti ali tehnike rečejo v Ameriki doktorat filozofije). Prispevki v angleščini so koristni vsaj iz dveh razlogov: zaradi možnosti neposredne izmenjave prispevkov z znanci, prijatelji in priložnostnimi interesenti v tujini (recimo kar od Amerike do Japonske) in zaradi obvladovanja (tudi učenja) jezika, ki je nujnost pri pripravi prispevkov (tudi tistih, ki izhajajo v Informatici) za objavo v tujih časopisih. Časopis Informatica sodi torej v "nacionalno ligo" jugoslovanskih strokovnih časopisov, kar je bilo tudi povsem jasno ob njegovi ustanovitvi. Povzetki v japonščini so bili res maloštevilni (izjemni) in so bili predvsem kulturni preizkus možnosti japonskega zapisa na slovenskih tleh. V industriji že danes poudarjamo nujnost obvladovanja japonščine, ki se nam za naš razvoj kaže enako pomembno kot obvladovanje angleščine. Informatica prinaša tudi znanstvene prispevke, ki se financirajo v obliki raziskav iz znanstvenih družbenih skladov. V tem svojem segmentu je Informatica znanstveni časopis, četudi tretjerazreden, kot ga lahko pač oblikujejo domači znanstveniki. Na koncu tega dolgega odgovarjanja lahko odgovorim zaradi meni vsiljene konverzacije le v slogu "milo sa drago": tudi med podpisniki pisma ne vidim nobenega posebno uglednega domačega ali tujega znanstvenika in tako naprej. Za to nekorektnost se upravičujem, vendar piscem kaj boljšega ne znam povedati. Seveda bi bilo zanimivo spremljati odmevnost člankov v Informatici. Verjetno bi za to uslugo lahko zaprosili VINITI (Vsezvezni inštitut za znanstveno informacijo) v Moskvi, ki mu časopis redno pošiljamo.

2. Recenzija prispevkov v časopisu je za naše razmere normalna, čeprav bi lahko bila strožja pa tudi mednarodna in z višjo pedagoško stopnjo (tudi ob večji angažiranosti podpisnikov). Normalni sistem recenzije bi lahko deloval s podpisanimi formularji treh neodvisnih (po posameznih lobijih ločenih) recenzentov, kjer bi moral vsak recenzent dokazovati predvsem bolj, zakaj članek podpira, kot pa, zakaj ga zavrača. Glavni urednik bi lahko bil le razsodnik v primeru nasprotujočih mnenj recenzentov. Recenzija prispevkov v Informatici je opravljena navadno na mestu njihovega nastajanja, dajejo jo mentorji, navadno po konzultaciji z glavnim urednikom. Zavrnitev prispevkov je vselej spremljana s pomočjo za njihovo dodelavo. Razmerje takih prispevkov (od začetne do končne faze) znese vsaj 30%. Problem tehničnega odbora je bila de facto njegova samoukinitev, saj so se v tem razdobju pojavili novi mentorji in sodelavci, ki so bili v časopisu aktivni. Tehnični in uredniški odbor sta bila imenovana v začetku leta 1977 po tedaj veljavnem teritorialnem in strokovnem "ravnotežju" in seveda s ciljno strategijo, da je potrebno oblikovati zadosten krog pisočih v okviru časopisa. Lektoriranje člankov je dolžnost samih avtorjev, kar je bilo tudi razumljivo v dobi proizvodnje prispevkov s pisalnim strojem in v obliki, v kateri se prispevki objavljajo (fotokopiranje). Za terminologijo je v okviru tehničnega odbora (vendar ne recenzije) skrbel prof. dr. L. Pipan, za angleščino pa avtorji oziroma njihovi lektorji angleščine.

Odgovor, ki zadeva t.i. glavne razloge za pisanje pisma piscev, bo v tem preliminarnem odgovoru urednika karseda kratek.

a) Za to izjavo, ki je iz teksta sicer iztrgana, stojim brez pridržkov. To potrjuje danes že praksa nekaterih komercialnih izdelkov (glej npr. predavanja prof. B. Souček v okviru SAZU, MIPRO '88 in njegovo knjigo o nevralnih računalnikih, ki je nedavno tega izšla pri založbi J. Wiley).

b), c) in d) Ti citati se nanašajo na podvig, ki sem ga imenoval ekspedicija Parsys. Verjetno moti pisce pisma najbolj to, da je projekt Parsys tudi za strokovno javnost osamljen primer, ki se ni rodil v univerzitetnem in inštitutskem okolju. V industriji gledamo na slovensko računalniško znanost morda neobjektivno kot na posebno obliko počivanja in spavanja. Če je pismo piscev tudi odločenost, da s tem prekinjajo, potem to pismo vsekakor pozdravljamo.

e) Da nastaja danes "nova matematika" v različnih oblikah po svetu, ni nobena skrivnost. Te "matematike", ki si nadevajo različna imena, seveda še niso znanje, ki bi bilo zrelo za sprejemanje v matematične katekizme. Razvoj enostavno kaže, da so minili časi t.i. izključno trdne matematične doktrine in da postaja matematika kot disciplina občutljiva tudi za mehkejši konceptualizem (npr. adaptivna, fuzzy matematika in cela vrsta aksiomatiziranih logik kot so že stara modalna logika, pa logika prepričanja, zavesti, umovanja, inteligence, informacije itd., ki trasirajo prav pot uporabe nevralnih računalnikov oziroma se priznavajo kot smiselne in zdravorazumske formalizacije v okviru umetne inteligence, kognitivne znanosti, filozofije). Vse to se seveda dogaja brez

blagoslova pravoverne matematike, vendar prav v teh raziskavah sodelujejo tudi matematiki in seveda industrija (npr. IBM). Informatika pač ni matematika, kar je menda razumljivo. O teh poskusih bomo lahko zapisali več podrobnosti v naslednjih prispevkih Informatice.

f) Kaj je v citiranem stavku narobe, bi morali izpostaviti predvsem pisci, ne pa le kazati nanj, kako je sporen. Da ta stavek ne more potrjevati prepričanja in tudi ne apriorističnega pudarjanja nepreklicne znanstvenosti piscev, je tudi razumljivo.

Glede stila nekaterih, prav gotovo ne vseh prispevkov, vključno s pismom piscev, pa se s pisci niti približno ne bi mogel strinjati. Zato naj mi bo v posebnem prispevku dovoljeno citiranje neobičajnih, fantastičnih (to je grda slovenska beseda, ki zveni kot psovka), laži-znanstvenih in neznanstvenih izjav v najuglednejših mednarodnih znanstvenih in strokovnih revijah s področja računalništva in informatike.

3. Na to vprašanje iz pisma je bilo odgovorjeno že na več načinov. Poudarjam, da sem v pismu piscev iskal tudi znanstveno razpravo in ne le namigovanja, ki nimajo z znanostjo in trditvijo, kako so pisci znanstveni, ničesar skupnega.

Pismo podpisnikov razumem predvsem kot njihovo pripravljenost, da v slovenskem, jugoslovanskem in mednarodnem znanstvenem prostoru dajo svoje znanje in svojo znanost v objavo in v sodelovanje tudi v časopisu Informatica. Menim, da bi s tem časopis Informatica postal tudi lažje znanstveni tržni produkt. Razmere so vsekakor dozorele: to pa ni samo redefinicija neke politike, temveč je spremmeba vedenja in vednosti akterjev, ki s svojim delom in sodelovanjem lahko proizvajajo ustreznejši časopisni produkt.

Urednik

SE O SONCIH V PRAZNEM PROSTORU

Anton P. Železnikar, Iskra Delta

Veliko sonc kroži v praznem prostoru: vsemu, kar je temnega, govorijo s svojo svetlobo - meni molčijo.

O, to je sovraštvo luči nasproti svetečemu, neusmiljeno hodi svoja pota.

Na dnu srca je krivično nasproti svetečemu: mrzlo nasproti soncem - tako se giblje vsako sonce po svoji poti.

Friedrich Nietzsche [1] 112

O ločevanju znanosti in filozofije

To, kar posebej "odlikuje" "današnjega človeka" kot človeka znanosti, je, da ne ve o sebi ne kdo ne kako JE in da se s to svojo nevednostjo niti ne sreča kot z zadevo svoje lastne človeškosti.

Ivan Urbančič [2] 428

Da znanstvenik ni filozof in filozof ni znanstvenik, to bi lahko bila resnica predvsem domače (izvirne), naravoslovno pravoverne utopije. Seveda, če bi bilo mogoče ukinjati prepletenost kulturne informacije (filozofije, znanosti, tehnologije, zdravega razuma) in povezanost informacije znanosti (npr. interdisciplinarnosti umetne inteligence: psihologija, kognitivne znanosti, neurofilozofija, modalna logika itd.). V tem domačem dialogu s kritiki o škodljivosti ali neprimernosti filozofsko-znanstvene relacije (za znanost) se ne bi skliceval na avtoritete iz matematično-filozofskega sveta, npr. na Descartesa, Pascala, Lebniza, Whiteheada, Hilberta, Russla, Wittgensteina itd. ali na kritična stališča samih akterjev umetne inteligence, kot so npr. D. McDermott (Artificial intelligence meets natural stupidity [6]), Winograd, Flores [3], H. L. Dreyfus, S. E. Dreyfus [4], kritika in odgovori na kritiko v časopisu Artificial Intelligence [5], vrsta kritičnih prispevkov v Mind Design [6] itd. O teh in podobnih izhodiščih sem pisal tudi sam [7].

Sonce informacije - če uporabim Nietzschejevo prispodobo - ima svojo pot, ki ni prema (platonična) steza matematike, šolniške podatkovne strukture, umetnointeligenčne (ljubljanske) subkulture, dorečene in varne znanosti in njenih v njih zaverovanih uživalcev. Matematika se je varno in seveda skladno z mejami svojih možnosti ognila bistvenim problemom informacije, čeprav je nekakšno merjenje (količino) informacije razglasila za informacijsko teorijo. Sicer pa je Dretske [8] dal dovolj objektivno in umirjeno kritiko te skrajno zožene in poenostavljene discipline (glej nekatere teh značilnih citatov v [14]).

Ne domišljam si, da sem v svojih delih [9, 10, 11, 12, 13, 14] sploh zapisal kaj izredno izvirnega o informaciji, kar bi bilo izven obstoječega območja zdravega razuma, ki že stoletja govori o pojavu, ki ga razume kot informacijo, čeprav mu nadeva različna imena (npr. tudi pomen, bistvo, bit, eksistenca, pojem itd.). Kibernetsko pojmovanje informacije v živem in tehnološkem je vsaj od Wienerja dalje to, kar je predmet mojih spisov. Kot živo bitje pa nosim prepričanje, da je te informacijske principe mogoče simbolično formalizirati in strniti v logično konstrukcijo tako, da bi ta lahko postala predmet uporabe zmogljivejših in novih strojev, kot so npr. t.i. nevralni ali paralelno masivni računalniki. To pa seveda ne pomeni, da postopek simbolične formalizacije informacijskih principov ne bo proces nastajanja, ki se bo razvijal skozi prihajajoče obdobje z domiselnostjo in iskanjem novega in predvsem s povezovanjem filozofskega in znanstvenega, konceptualnega in konstruktibilnega.

Tudi si ne domišljam, da bo termin informacijska logika v okviru redefinicije informacije, kot jo predlagam, preživel. Verjetno se bo našel nov pojem za nekaj, kar je v bistvu postinformacijsko. Že danes se slišijo glasovi, da je doba informacijskega (seveda na temeljih starega razumevanja informacije kot nekakšnega pretoka znanja) v zatonu in da prihaja doba umskega (mind-like), ki naj bi prekvasilo ne le znanost in tehnologijo, temveč tudi ekonomiko in socialno filozofijo in akcijo.

O citiranju in kritiki citatov

Ideologikritika ni mišljenje, a si le tako zagotavlja svojo premoč nad vso filozofijo in tudi ideologijo.

Ivan Urbančič [2] 428

Citiranje uporabljam navadno kot vodilo oziroma vzpodbujanje, ki se določenemu miselnemu toku prilega in mu daje metaforično usmeritev. Seveda pa je citiranje mogoče uporabiti tudi kot napad na določena stališča in izjave. Vendar je v tem primeru smiselno in civilno kulturno upoštevati kritično toleranco predvsem takrat, kadar so informacijska ozadja izjavljavcev in kritikov preveč oddaljena in se

zaradi tega pojavlja problem razumevanja kritiziranega.

V citatu (a) [0], v katerem se mi očita filozofska spekulacija [15], gre za opozorilo študentom, da v okviru nekaterih današnjih računalniških predmetov (operacijski sistemi in prevajalniki) ne bo mogoče razreševati problemov inteligentne uporabe strojev. Opozorilo je pedagoško in je izziv študentu, da razvija potrebno kritičnost do zmogljivosti današnjih algoritmičnih konceptov, ki obvladujejo in omejujejo današnjo računalniško uporabo. Stvar osebnega okusa je, kako je mogoče študenta motivirati in ga pripravljati na problematičnost njegove poklicne dejavnosti, ki bo drugačna od pričakovane. Morda pa je vendarle simptomatično, da študentje, na njim lasten način, demonstrirajo bojkot (tihi štrajk) predavanj, v katerih se zrcali strokovna in pedagoška nesposobnost učitelja, ki zmore le pomanjkljivo interpretirati "znanost", ki ni predmet njegovega lastnega raziskovalnega dela in je le mehanično in pedagoško pomanjkljivo prenašanje znanega pa tudi nerelevantnega. Ob tem bi se morali zamisliti prav učiteljevalski podpisniki pisma, ker ta ugotovitev velja tudi za njih same. Seveda ne bi imel posebnih zadržkov do objavljanja nekaterih mojih spisov tudi v filozofskih časopisih, čeprav sem prepričan, da v njih ne gre za "filozofsko" filozofijo.

Citat (e) [0] je programska izjava o možnostih nastajanja informacijske logike. Ozadje te logike je podano z možnostjo informacijskih principov, kot sem jih formuliral v [12]. Tu ne gre za platonsko izjavo, saj bi bilo v (bistveno) omejeni obliki mogoče formulirati tovrstno logiko tudi s pomočjo navadnega matematičnologičnega konstruiranja (npr. s posiljevanjem formalnega jezika v okvire predikativne logike prve stopnje). Da pa rojevanje take logike ne bo enostavno in matematično rutinsko, opozarjam implicitno in eksplicitno prav s filozofijo določenih pojmov. Področja informatike in njenih problemov nisem nikoli razumeval kot trdne, algoritmične discipline, čeprav računalniški "strokovnjaki" (v stilu objektov iz razreda "Fachidiot") prepričujejo sami sebe, da je informacijsko (tudi inteligenčno) v bistvu prav in samo njihova racionalistična znanost.

O histeričnem pamfletiranju

Znanost danes ne misli: in samo v tem je njena moč, učinkovitost.

Ivan Urbančič [2] 428

Menim, da pisci pisma iskreno verjamejo v to, kar so zapisali. Odstavek, ki govori o ponovni objavi prispevkov, pa bo - upam na iskreno veselje piscev pisma - upoštevan. Seveda ne gre za ponovno objavo kar vsega povprek, saj je npr. potopis [16] le pričevanje o določeni izkušnji projekta paralelnega računalnika. Po srbskem reklu bi lahko prostodušno ugotovil, da se nekoliko histerično mešajo babe in žabe. Za objektivno komunikacijo bi bilo potrebno odprto pismo prevesti v angleščino - in to bi lahko storili pisci sami. Ker domači znanosti ne velja zaupati, naj bi tudi za pisce pisma veljalo le to, kar bo pripravljen potrditi nek zunanji klan. Upam, da se bo to zares zgodilo!

Ne morem si kaj, da odprtega pisma ne bi razumel kot histerično pamfletiranje, ki dosega svoj vrh z Nobelovim nagrajevanjem. Ali se morda motim, če govorim o čustveni razdražljivosti piscev pisma, njihovi nagnjenosti k blasfemiji, posebni avtosugestiji in o njihovih motivativnih konfliktih? Ali ne kaže "napad povprek" na dramatično obliko vzklikanja in celo neprikrite agresije, ki je daleč od kulturnega ali vsaj znanstveno sprejemljivega dialoga, za katerega je potrebna prisotnost publike, čeprav seveda "kritika" ni nevarna za okolico? Mislim, da bi pisci pisma svoj problem veliko lažje razrešili s svojim lastnim znanstvenim delom, ki bi ga z veseljem objavljali v Informatici in spremljali njihovo uspešnost v tujini. Tako pa se mi njihova intenzivna čustva kažejo predvsem kot simptom nemoči, nedozorelosti in žal tudi govorjenja v tropu (organicizma).

O rotenju, zaklinjanju in herostratstvu

Znanstveno rotenje in zaklinjanje ne zaležeta, če sta predmet in metoda rotenja in zaklinjanja deklarativno znanstvena. Naj mi bo dovoljen tale citat [17], ki se prav gotovo lepo prilega formalnemu vzklikanju piscev pisma:

Tine: *Moja metoda je metoda znanosti.*

Igor: *To so visokodoneče besede! Toda ali ne vidiš, da uporabljaš krožno dokazovanje: metoda ti pove, kaj je znanost, le-ta pa potrjuje metodo. V formalni logiki nisi ravno doma, ali ne?*

Pa tudi po motivacijski in logični plati ne verjamem, da je pri določenih obratih v mišljenju in tudi v znanosti herostratstvo primerna metoda, da se prav z njim blokirata napredek in razvoj disciplin, kot so npr. t.i. discipline umetne, nevralne in molekularne inteligence (ta klasifikacija je povzeta prosto po B. Souček). Informatika kot disciplina verjetno še ni dosegla svojega zenita, čeprav se ji danes že očita, da je zastarela in da ne dosega tega, kar se imenuje inteligenca oziroma um. Sam sem seveda toliko konzervativen, da trdim, da sta inteligenca in um kot živi procesnosti tudi informacijski obliki. Kakor koli že, politika celotne znanosti razvitega sveta kaže, kako bo ta prej ko slej postala del strategije, ki jo predsednik ZDA Ronald Reagan imenuje economy of mind. Kje smo v znanstvenem razmišljanju o trendih tega razvoja pri nas, na univerzah, v inštitutih in v industriji? Ali je sploh mogoče znanstveno raziskovati brez določene filozofije, ki vselej nastaja izven čiste znanosti in izven dosega mišljenja znanosti predanih in le njej vzklikajočih služabnikov? Ker tudi res nismo daleč od tega, kar pravi švedski matematik Axel Berringson: "Nesmiselno je iskati zabavo v matematiki v trenutku, ko je pol človeštva lačnega."

Slovstvo

[0] Odprto pismo (T. Pisanski etc.), Informatica 12 (1988) št. 3, (okviru te rubrike).

[1] F. Nietzsche, Tako je dejal Zaratustra, Slovenska matica, Ljubljana 1974.

[2] I. Urbančič, Mišljenje - pesnenje, Nova revija 7 (1988) št. 71-72.

[3] T. Winograd, F. Flores, Understanding Computers and Cognition, Ablex Publ. Corp. Norwood, NJ (1986).

[4] H.L. Dreyfus, S.E. Dreyfus, Mind over Machine, The Free Press, Mcmillan, New York, 1986.

[5] T. Winograd, A Response to the Reviews, Artificial Intelligence 31 (1987) 250-261.

[6] J. Haugeland (Editor), Mind Design, The MIT Press, Cambridge, Mass 1981.

[7] A.P.Zeleznikar, Artificial Intelligence Experiences Its Own Blindness, Informatica 11 (1987) No. 3, 25-28.

[8] Dretske, Knowledge and the Flow of Information, Basil Blackwell Publisher, Oxford 1981.

[9] A.P. Zeleznikar, Na poti k informaciji, Informatica 11 (1987) št. 1, 4-18.

[10] A.P. Zeleznikar, Information Determinations I, Informatica 11 (1987) št. 2, 3-17.

[11] A.P. Zeleznikar, Raziskave računalnikov in informacije v naslednjem desetletju, Informatica 11 (1987) št. 2, 57-59.

[12] A.P. Zeleznikar, Principles of Information, Informatica 11 (1987) št. 3, 9-17.

[13] A.P. Zeleznikar, Information Determinations II, Informatica 11 (1987) št. 4, 8-25.

[14] A.P. Zeleznikar, Problems of the Rational Understanding of Information, Informatica 12 (1988), št. 2, 31-46.

[15] A.P. Zeleznikar, Uvodno predavanje, Informatica 12 (1988), št. 1, 3-5.

[16] A.P. Zeleznikar, Parsys Expeditions to New Worlds II, Informatica 12 (1988), št. 1, 77-91.

[17] V. Pečjak, Nastajanje psihologije, Dopisna delavska univerza Univerzum, Ljubljana 1983.

Prateći razvoj novih kompjuteriziranih tehnologija i postupaka za proračunavanje, projektiranje i oblikovanje, te poslovne i proizvodne sisteme kao i napredak u teoriji informacijskih, plansko-poslovnih i proizvodnih sistema, odlučili smo da za ovaj, jubilarni, 10. međunarodni simpozij «Projektiranje i proizvodnja podržani računalom – PPPR '89«, koji će se održati od **17–19. 10. 1989**, bitno proširimo i drugačije grupiramo teme Simpozija.
Naime, suvremeni pristup sve više zahtijeva integraciju i sistemska znanja, što istovremeno nameće napuštanje podjele po užim strukama. Teme bi pored CAD, CAM, CIM i CAE trebale zahvatiti i u područje ekspertnih sistema, umjetne inteligencije, problematike velikih sistema, dajući i pregled postojećeg stanja i predvidivi bliski razvoj preko ovih novih područja. To su i razlozi zbog kojih je Programski odbor zamijenio dosadašnje sekcije zasnovane na strukama (elektrotehnika, strojarstvo, brodogradnja, građevinarstvo itd) novim temama, koje obrađuju računarske postupke primijenjene na probeme zajedničke različitim strukama.

TEME SIMPOZIJA

1. **Metode proračuna deformabilnih tijela**
   (elastičnih, plastičnih, viskoelastičnih, viskoplastičnih itd)
2. **Metode proračuna polja**
   (elektromagnetskih, toplinskih, hidromehaničkih, aeromehaničkih, optičkih itd)
3. **Projektiranje informacijskih sistema**
   (tehničkih, poslovno-planskih, zdravstvenih itd)
4. **Operacije i proizvodni procesi**
   (metode analize, sinteze, simulacije i optimiranja kontinuiranih i diskretnih procesa u tehničkim sistemima različitih struka)
5. **Oblikovanje računalom**
   (oblikovanje proizvoda na danoj ili pretpostavljenoj osnovnoj morfološkoj, topološkoj i geometrijskoj strukturi, koja proizlazi rješavanjem primarnih (funkcijskih) i sekundarnih (analize) procesa. To se završno oblikovanje temelji na kriterijima funkcionalnosti, ergonomičnosti, estetičnosti itd)
6. **Novi horizonti u primjeni računara**
   (koncept ekspertnih sistema, nove metode razvoja software-a, komunikacija između računara i korisnika, vrlo kompleksno projekiranje, metode upravljanja aplikacijama, utjecaj novih metoda u grafici i novih grafičkih mogućnosti, utjecaj novih jezika i očekivanih standarda)

ROKOVI PRIJAVE SUDJELOVANJA S RADOM

Ispunjenu prijavu zajedno sa sažetkom od oko 500 riječi, koji dobro ilustrira sadržaj i svrhu rada poslati najkasnije do **1. 11. 1988.** godine na adresu tehničkog organizatora – ATLAS, Kongresni odjel Zagreb.

Odabrane radove, napisane prema uputama koje će autori dobiti uz obavijest o uvjetnom prihvatu rada na temelju sažetka treba poslati predvidivo do **3. 4. 1989.** Predavačima će prilikom izlaganja biti na raspolaganju diaprojektor, grafoskop, videoprojektor (VHS sistem).

Jedan autor može prijaviti najviše dva rada, bilo kao autor ili kao koautor. Obavezni, sastavni dio prijave rada je anketa o radu tiskana na poleđini prijave. U obzir za tisak će se uzimati samo radovi autora za koje bude pravovremeno uplaćena kotizacija.

KOTIZACIJA

Informacije o visini kotizacije i načinu uplaćivanja biti će objavljene u 2. obavijesti (predvidivo svibanj 1989. godine).

MJESTO I VRIJEME ODRŽAVANJA

Simpozij će se održati u Kongresnom centru Zagrebačkog velesajma, Avenija Borisa Kidriča 2, Zagreb, u okviru specijalizirane priredbe INTERBIRO-INFORMATIKA od **17–19. 10. 1989.** godine.

STRUKTURA SIMPOZIJA

Usmena izlaganja, poster sekcije, prezentacije hardware-a i software-a, izložbe.

JEZICI

Službeni jezici Simpozija su hrvatskosrpski i engleski jezik uz simultano prevođenje. Zbog međunarodnog karaktera Simpozija, mogućnosti bolje komunikacije i mogućeg objavljivanja u stranim časopisima posebno preporučamo autorima da radove pišu na engleskom jeziku.

HOTELSKI SMJEŠTAJ

Za sudionike Simpozija organizator će osigurati smještaj u hotelima A i B kategorije. Sve detaljne informacije kao i način vršenja rezervacije smještaja biti će objavljene u 2. obavijesti.

PREDSJEDNIŠTVO ORGANIZACIJSKOG I PROGRAMSKOG ODBORA SIMPOZIJA

Predsjednik
Dr Zijad Haznadar, redovni profesor Elektrotehničkog fakulteta Sveučilišta u Zagrebu

Potpredsjednik
Dr Vesna Jurčec, savjetnik Republičkog hidrometeorološkog zavoda SR Hrvatske, Zagreb

Tajnici
Mr Sead Berberović, asistent na Elektrotehničkom fakultetu Sveučilišta u Zagrebu

Dubravka Pavlović, ATLAS, putnička agencija, Kongresni odjel Zagreb

ADRESA TEHNIČKOG ORGANIZATORA

ATLAS – Kongresni odjel
Trg senjskih uskoka 7
41020 Zagreb
Telefon: 041/525-333, 528-094
Telex: 22413 atlcon yu
Telefax: 041/525-468