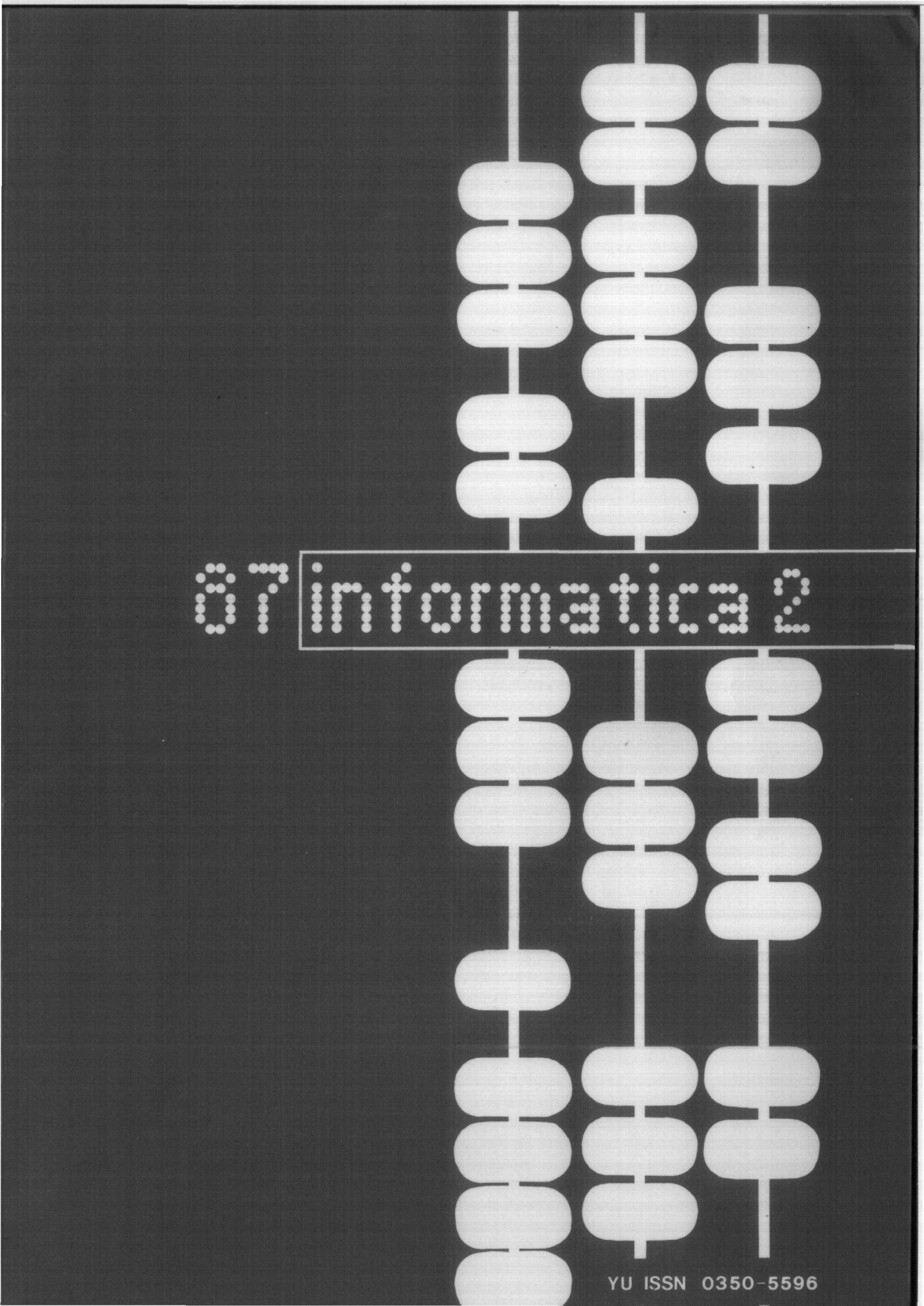


YU ISSN 0350-5596

AVTORJEM CASOPISA INFORMATICA

Zaradi možnega plasmaja časopisa Informatica v tujini naprošamo avtorje časopisa Informatica, da pišejo v prihodnje svoje članke, referate, eseje in novice po možnosti v angleščini. Povzetek v domačem jeziku z naslovom prispevka naj bo dodan na posebnem listu v formatu stolpca (47 znakov na vrstico).

AUTORIMA CASOPISA INFORMATICA

Zbog mogućeg plasmana časopisa Informatica u inostanstvu, autori časopisa Informatica se umoljavaju da pišu u buduće svoje članke, referate, eseje i novosti po mogućnosti na engleskom. Abstrakt u domaćem jeziku sa naslovom članka neka bude dodat na posebnom listu i izpisan u formatu stupca (47 znakova na redak).

TO AUTHORS OF INFORMATICA

To achieve the possible exchange of Informatica with foreign journals, the authors of articles for Informatica are kindly requested to submit their articles, papers, essays, and news in English, when possible. Abstracts in a domestic language containing the title of an article should be submitted on a separate sheet and printed in the obvious column format (47 characters per line).

informatica

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

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

Headquarters:

Informatica, Parmova 41, 61000 Ljubljana, Yugoslavia. Phone: 61 31 29 88. Telex: 31366 yu delta

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

Printed by: Tiskarna Kresija, Ljubljana

JOURNAL OF COMPUTING AND INFORMATICS

YU ISSN 0350-5596

VOLUME 11, 1987 – No. 2

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 11, 1987 – ŠT. 2

Uredništvo in uprava:

Informatika, Parmova 41, 61000 Ljubljana, telefon (061) 312 988; teleks 31366 YU Delta.

Letna naročnina za delovne organizacije znaša 11990 din, za zasebne naročnike 2990 din, za studente 990 din; posamezna številka 4000 din.

Številka žiro računa: 50101-678-51841

Pri financiranju časopisa sodeluje Raziskovalna skupnost Slovenije

V S E B I N A

UDK 519.724

Anton P. Železnikar
ISKRA DELTA, Ljubljana

Abstract. This essay is the continuation of the previously published 'On the Way to Information' (Zeleznikar, OWI), which represents a framework for the program of informational investigation. Even though defining determinations of information is not a thankful task, it has to be done in order to attain the necessary foundation upon which a more exhaustive investigation or even a soft formalization of informational comprehension will become possible. This essay deals with the following topics: a soft introduction into the phenomenology of information; a very basic question of information; the meaning of the term 'information'; meanings of the direct derivatives of the term 'information'; the meaning of the verb 'inform', of the adjective 'informative', of the noun 'informatics', of the verb 'informatize', of the noun 'informatization', and of their various derivatives; informational forms and informational processes; information and counter-information; informational recurrence; forms and processes of life; intelligence; cultural forms; a being's information; Being and time; phenomenology; metaphysics and ontology; autopoiesis; neural science and information; and brain and behavior. This essay represents the first part of basic information determinations or of basic information investigations. In the second part of the essay various cultural forms will be investigated from the informatical point of view.

要旨. このエッセイは「情報への道」(Zeleznikar, OWI)という論文の続編である. 同論文では情報及びその関連分野の研究プログラムが論じられた. 情報という概念を捉えることは容易ではないにも関わらず, 情報の理解にあらゆる側面から光を当てるための考察 (それはインフォーマルな記号化も含みうる) の欠かせない基礎作りが必要である. よって, このエッセイでは次のようなトピックを取り扱う. すなわち: 情報現象論へのインフォーマルな入門; 情報を巡る基本的な諸問題; 情報という用語の意味; 情報'information'という用語から直接派生される諸用語の意味; 'informatics'という名詞, 'informatize'という動詞, 'informatization'という名詞及びこれらの用語から派生されうるものの意味; 情報的形態と情報的過程; 情報と反・情報; 情報的再帰性; 生命の形態及び過程; 知能; 文化における諸形態; 生命体の情報; 時間と生命体; 現象論; 形而上学及び存在論; autopoiesis; neural-science と情報; 脳と行動. 当エッセイは情報及びその周辺の限定を目標とする考察の第一部である. 第二部ではいくつかの文化における形態を情報的観点から論じる予定.

Keywords. Allopoietic, appropriation, arising, artificial intelligence, autonomous, autonomy, autopoiesis, behavior, Being, being, being-generative, being-in-the-form, being-in-the-form-as-a-form, being-in-the-form-as-a-process, being-in-the-information, being-in-the-process, being-in-the-process-as-a-form, being-in-the-process-as-a-process, being-in-the-world, being-parallel, being-recurrent, being's information, biologic machine, blindness, breakdown, brain, central nervous system, cerebral hemisphere, circular, circularity, cognition, cognitive psychology, cognitive science, coming into existence, communication, computer science, cortex, cortical lobe, counter-information, counter-informing, cultural form, culture, DNA, eidos, enzyme, environment, evolution, filter, form, form of life, forma, formatio, formare, gene, gene mutation, generative, genetic information, glia cell, heteropoietic, immune system, inform, informable, informare, informateism, informatical, informaticable, informaticity, informaticness, informatics, information, information arising, information investigation, information machine, information organization, information philosophy, information processing, information science, information structure, information system, information technology, information theory, informationable, informational, informational form, informational function, informational hierarchy, informational imagination, informational noise, informational process, informational recurrence, informational reflection, informationalism, informationality, informationally

parallel, informationness, informationism, informatism, informativable, informative, informativism, informativity, informativness, informatizability, informatizable, informatization, informatizationable, informatizational, informatizationess, informatizationism, informatize, informatizing, informatizeness, informic, informing, informingness, informism, informistic, informity, informness, intelligence, intelligent, intelligent illusion, intelligent stupidity, interrogated, language, learning, life, living being, living cell, matter, meaning, mentation, metaphysics, mind, modulation, molecular receptor, molecular transmitter, molecule of life, morfe, nerve cell, neural science, neural system, neuron, ontology, organization, parallel processing, parallelism, perception, perturbation, phenomenology, phenomenon, philosophy, poiesis, population, process, process of life, protein, question, questioning, realm of information, recurrence, recurrent, replication, representation, reproduction, RNA, selection, self, self-production, semantics, sensory information, sociology, structure, subjective, subjectivity, system, time, and understanding.

0

> "... Science does not think in the sense in which thinkers think. Still it does not at all follow that thinking need pay no attention to the sciences. ... And since science does not think, thinking must in its present situation give to the sciences that searching attention which they are incapable of giving to themselves."
>
> (Heidegger, WCT, 134, 135)

This essay can be read and comprehended by anyone who is willing to make a philosophical endeavour in thinking through the proposed views of information. Intuitively, the notion of information is well-known from the colloquial grasping of information-relevant and information-related subjects and objects. A being is experiencing information and informational processes in everyday life. This experience is coming into existence as mind is looking into itself, as mind is producing mind, and as the produced mind is becoming the producer. For this purpose of understanding, no particularly advanced knowledge is needed, however, careful reflection, imagination, and recursive thinking is necessary. In the case of information and of information-related problems, being is beginning to think a new way cognitively, constructively, and diversely. Hence, one only has to follow the way, which exhibits a new, varying, and motivating orientation.

In this essay a series of determinations of the essence of information will be elaborated upon and conjoined: information as mastery over itself; information as information and counter-information; information as Being and being; information as substance and process; information as form and matter; information as space and time; information as intelligence and machine; and information as philosophy and language. In the first part notions and concepts will be presented concerning some new linguistic derivatives of the known informational terms (etymological archetypes). These notions will be arising concepts, always coming into existence as partially developed subjects, modifying their nature and in this respect, informational by themselves.

Information investigations seem to become relevant not only in the contemporary context of neural science, artificial intelligence, new information philosophy, information science, and information technology, but also in information psychology (processing of information in human brain) or cognitive science (as understanding of cognition and cognition of understanding). There are some traditional objections and methodological, philosophical, and ideological doubts coming from sociology, from its dogmatically grounded component, arguing that information cannot be a universal means for understanding the physical, social, living, and non-living world. At the same time, informatics as a new, arising, scientific and technological discipline is calling for a more elaborated notion of its main subject - information. A new, sufficiently diverse, heterogeneous, disparate, and intelligibly enriched philosophy of information is needed to enable human progress in investigating and designing informationally related subjects and machines.

Information concerns commonplace experience, life processes, and behavior of living beings. Information is the most general investigation of everyday life, the most primitive, rudimentary, or sophisticated philosophy which delivers information for existential, life-concernig, and being's orientation in the future. This essay is not a systematic investigation of determinations in which notions concerning information are developed in a hard, rigid or rigorous way. At the beginning of a philosophy - and this holds true for the philosophy of information - philosophical determinations are soft, circular, hermeneutic, and clumsy. Systematic research of information philosophy does not exist yet, and therefore, must first come to the conscious surface on the basis of several determinations being formulated and understood in the new, informationally cognitive context.

At this stage of investigation one can search for constitutive pieces which are fundamental and foundationally conjoined determinations. The need of a solid arsenal of information determinations is the main motivation of this essay. This new series of determinations will include several new terms of information and will call - against conventional linguistic legality - for the introduction of new linguistic entities, i. e. terms and meaning of terms.

Philosophy is not a suitable, adequate, or genuine way for analyzing, discovering, and examining the essence of information because it represents only a particular, informationally limited form or process of information itself. Furthermore, philosophy is using linguistic

means, whereby language is another form or process of information confinement. The power of information in the human cortex, when information is used for information investigation, is much more complex than linguistic or philosophical expression, representation, cognition, and understanding could ever be. The most proper system for information investigation is the total being's information, which is coming into existence solely in a being itself, into his own information system during his life. This information concerning information is a being's information philosophy, which is a part of a being's ontology and of a being's total information. In this context, the culture of a being's population is an external information, not necessarily dwelling in the being, which is incidentally rushed, externally archivated, traditionally oral, and orthodoxly written, general information. So, it is becoming clear, that an arising information philosophy can speak about its objects and subjects solely from its own, informationally narrowed view.

1

Information is not only
the instantaneous-informational
in information itself;
it is also information
coming into existence
concerning this information.

In this section the archetype 'information' and its derivatives will be examined in a etymological, comprehensive, and sequential manner. Thus, the following derivative sequence of terms will be loosely determined:

information: informational, informationism, informationalism, informationness, informationality, informationable.

1.1. Question of Information

The question of information is the most fundamental and general one. Any question produces information which comes into existence as a result of questioning. This result is the interrogated. The question of information, its questioning, and its interrogated are all in the domain of information. The question of information is a part of information itself and, by this, is informationally transparent or informationally recurrent. Even though the question of information is informationally general, possibly the most general, it is still senseful and particular in this context.

Questioning about information is a cognitive seeking of information by information alone; it is an informational investigation of information. The question in this questioning is information, however information is also questioning about the interrogated. simultaneously, a question of information is an informational resolution by questioning. Question, questioning, and the interrogated (as a result of questioning by question, i. e. the requested) are informationally joined and informationally dependent: all these are recurrent information about information. The interrogated is only an instantaneous information result, only the instantaneously, informationally investigated. The goal of informational questioning about information is the temporarily informational investigations about information. At this point, developing information is questioning about information alone and thus, it is developing information itself as its own object - the interrogated - and as its own questioning. This questioning about information by information, where questioning is coming into existence as information, is transparent in information itself or more precisely, recurrent and generative in the realm of questioning about information.

The essence of information is the arising of information about information. In this context, it is also possible to investigate the structure of questioning about information.

As a seeking, questioning needs previous guidance from what it seeks (Heidegger, BW, 45).* Considering this formula, information about information, which is sought and represents the interrogated, does already exist to some extent and is moving in the direction of understanding cognition of information and in the direction of informational cognitive understanding of information, which is already coming into existence. From this understanding, questioning about information, information about the essence of information, and the tendency towards informational understanding of information are all coming into existence. With the question of information, the understanding of information is coming into existence. Certainly, the final request can never be expected, because the interrogated is a continuous arising of cognitive grasping of information. From the informational point of view, every concept is information coming into existence, so that the grasping of a notion is nothing else than an arising information or an informing of information. This information arising is a fact coming into existence independently from the realm of grasping or cognitive understanding.

Information is circular concerning information. Information is the foundation which constitutes the understanding of Being and time; in this respect it is a being's information. Information is everything that can be imagined in regard to an object or a subject, a form or a process, a perturbation or an information, etc. Information is, for instance, representation, understanding, cognition, meaning, object, process, being, everything imaginable, perceiving, existing, non-existing, and coming into existence. If nothing exists, this is also information. In a similar sense, matter, energy, space and time are only information. If something is not information, its representation of non-existence in the realm of information is that it does not exist.

Information coming into existence through this research of information is contributing to the understanding of information. If everything is

---

* To Heidegger's formula of questioning the so-called continuing principle of Leibniz (AGB, Rieppel, 37) can be compared: "... Alle diese Formen sind in unsrem Geiste enthalten und bestaendig in ihm vorhanden, da der Geist stehts alle seine zukuenftigen Gedanken ausdrueckt und in verworrener Weise schon an alles denkt, was er jemals auf distinkte Weise denken wird. Nichts kann uns daher gelehrt werden, dessen Idee wir nicht schon zuvor im Geiste truegen, gleichsam als Materie, aus der der Gedanke sich gestaltet. ..."

information, then it is understandable that philosophy of information and philosophy in general are only particular, e. g. philosophical information. Thus, thinking, language, literature, science, art, music, ideology, speaking, behaving etc. are only particular forms or processes of information. Form and process are again information. This arising circularity of information is the basic information principle.

Cosmic and biological perturbations are information concerning processes of cosmic and biological changes, generations, and coming into existence. Since everything can be understood as a process, everything is generating and receiving information in some informational way. This philosophy of information can be expanded to living as well as non-living processes. For instance, a being's notion of Being is both partly cultural and socially interactive, and partly being-originated information.

The interrogated in the question of information is information. Information determines being in its whole, determines its life forms and life processes, and it also determines the entire cosmos and cosmic perturbations. If, in the context of information, the question arises what is being, the answer is the following: a being is determined only to the extent in which arising of a being's life (biological) information is coming into existence, where this information is meritory and decisive for a being's survival in a being's environment, for a being's reacting on external information processes, or for a being's behavior.

The extended notion of information will be developed throughout this essay. In information, the meanings of its linguistic derivatives as well as other informationally connected linguistic derivatives (substantival, verbal, adjectival, adverbial, pronominal, composite) will circularly come into existence and will circularly be included, nested, and opened for further extensions of their meanings.

1.2. Grasping the Meaning of the Term Information

As developed into a broader and more complex meaning, information needs an adequate and powerful language, which will enable a different orientation. From this new perspective the so-called informational tradition will come into existence.

The original meaning of the word information concerns form, form concerns vision, and vision concerns idea. Already in the antique philosophy the problem between matter and form arose. The question of form has always surpassed the question of matter. Being is not a raw matter, it is formed in some way, structured, and organized. Directly at this point it is possible to state the question: who is structuring and who is organizing this matter and in which way. The answer is: information, the forming itself, which is the process of forming substantial and processing patterns, being transferred from one generation to the next one, determining forms and processes of life. In this case, the biological is coming as close as possible to the informational.

Form is the essential root in grasping the meaning of information. The Greek 'eidos' has various meanings, e. g., face, figure, form, shape, appearance, aspect, beauty; idea, concept, thought, notion, property, kind; manner, way, mode, nature; state. Similarly, the Greek 'morfe' means form, shape, figure, body; grace, beauty. The Latin 'forma' (-ae, f.) is figure, form, appearance (exterior), beauty; picture, sign, plan, base, contour, manner, way, kind, quality. The Latin 'formatio' (-onis, f.) is creation, formation, representation (mapping). The Latin verb 'formare' has the meanings to form, shape, picture, represent; to educate, instruct, teach; to control, settle, adjust, regulate, put in order, make, do, create, produce, execute. The Latin prefix 'in' has spatial, temporal, and modal meanings. They are: about, after, against, as, at, behind, beyond, during, in, into, on, till, to, towards, until, up to, upon, within, etc.

In grasping the notion of information one has to consider the Latin word 'informatio' (-onis, f.), which is the composite of 'in', 'forma', and 'tio'. Here, the meanings of 'in' and 'forma', with all their diversities and combinations, have to be taken into account. Meanings of the Latin informatio are forming, putting into form, presentation, notion, description of notion, explanation, interpretation, etc. In philosophy, information has specific meanings. In scholasticism this meaning is etymological and ontological: it represents shaping of matter by form, where shaping is thought as a process itself and as a result. It has the meaning of forming the mind as a consequence of a co-natural object of cognition. Later, Descartes introduced the meaning of forming the consciousness by the physical structure of brain, which is becoming evident as perception. In this respect, the meaning of information as form and process in this essay does not reflect a very new or unusual meaning.

Today's meaning of information concerns communication or reception of knowledge or intelligence. But information is also knowledge obtained from investigation, study, or instruction, as well as data and facts. At last it is becoming clear in which direction the contemporary meaning of information is developing as form and process in the entire phenomenology and how this meaning will still arise and still come into existence during new investigations.

If information is an informational form or an informational process, what is the meaning of this form or this process? This question concerns the relation between form and its content (or meaning) or between process and its content (or meaning). Generally, the meaning of a form or a process is coded, hidden, and dependent on the observer, i. e., from the information, which handles form or process as information. This is the old and well-known problem of form and its content, where content of a form is the understanding attributed to this form. Generally, form is a two-component information, the first one belonging to the form and the second one to its content. Different forms may deliver equal contents to different observers. For example, different information may result in the same information through the different informing performed in different observers. On the other hand, a single form can deliver different information to different observes, i. e., a given information from an informational source can inform different information at different places (or for different purposes).

Being, when considered as a form, represents itself as information. However, a living being represents itself in an unlimited number of ways. Namely, in a changing environment, information in a being is unforeseeably arising

with a being's behavior. So, the form which is called being has its own various, changing contents. A very good example of form and meaning (content) is a being's language. Different living beings of a species understand language, which is spoken by an individual being, in different ways, getting different semantics from each of these ways.

Information literally has its historically earlier and later meaning of being-in-the-form. But, as grasping of form is being generalized, this meaning becomes being-in-the-process too. In the contemporary technological era this meaning is narrowed into it-gives, it-takes (message) and in philosophy into there-is, it-is.

For a living being, information means being-in-the-form and being-in-the-process. For contemporary philosophy, the main philosophical object remains information, which is Being-in-the-form (or beings-in-the-form) and Being-in-a-time (or beings-in-a-time) and where being-in-the-world (the German 'das In-der-Welt-sein') is a part of being-in-the-form and being-in-the-process.

Evidently, information means anything-being-in-the-form and anything-being-in-the-process. Furthermore, the meaning of information can be automatically extended into circular, recurrent, generative schemes like being-in-the-information, being-informed, being-informational, being-qua-information, etc. It appears, that in this philosophical and etymological context, the meaning of information was diversely and sufficiently enlightened by various views of thinking about the meaning of information.

Evidently, information has its informational structure and organization. Within these features it is a system, which produces informational arising, generating, varying, and coming into existence of this and other information. Information is a subject whose object is information. Simultaneously, information is its own arising product and arising producer, where the arising product incorporates the arising producer. From the processing and producing point of view, information is a system by itself. In this concern, information is self-producing, self-influencing, self-observing, self-referencing, self-arising, self-organizing, self-structuring, self-governing in the realm of information. However, information is also the non-self. Information is informationally systemic.

1.3. Informational

Here, several new terms, which are outside English linguistic legalities, are introduced. The meaning of the adjective 'informational' and the noun 'Informational' have to be determined. The terms informational and Informational and their derivatives will be used throughout the context of this essay.

The adjective 'informational' does not have the same meaning as the adjective 'informative'. Informational is the new adjectival derivation of information. Its meaning lies closer to 'information-like', 'information-equal', 'information-generative', 'information-circular', 'information-peculiar', 'information-characteristic', etc. In contrast, 'informative' has the usual and characteristically communicative meaning, which is similar to having, possessing, bringing, or carrying information. The meaning of 'informational' is evidently broader and concerns information more precisely than the meaning of 'informative'. Hence, the meaning of 'informative' is included in the meaning of 'informational'.

The meaning of the noun 'Informational' is constituted from the meaning of the adjective 'informational' and vice versa. Informational is everything concerning information and represents the complete realm of information. Thus, the meaning of Informational can be equated to the meanings of information-circular, Information-like, Information-equal, Information-characteristic etc. However, it also includes the meanings of information processing, informatics, information philosophy, etc. Thus, instead of 'information determinations' in the title of this essay, philologically more flexible and meaningfully appropriate would be 'informational determinations', which are determinations concerning the complete realm of Informational.

1.4. Informationism (or Informationalism)

If one attempts to understand everything as information (or Informational), to be an informational form or process, or to be a cosmic or a being's information, then this realm of understanding is called informationism (or informationalism). In this respect, informationism (or informationalism) can be viewed as a being's ontology or cultural ontology, as a particularly, information-like or informationally coloured metaphysics, modulated metaphysics or filtered metaphysics. Hence, informationism (or informationalism) in itself is nothing less than information (or Informational). If informationism (or informationalism) is a chosen way of thinking, it may develop and clarify a special and new understanding in the realm of information and Informational. The way of thinking in an information-like or informational manner is leading to a general philosophy of information. This way is satisfying the aim to develop a new philosophy circularly concerning information, Informational, and informationism (or informationalism) itself.

In this manner, informationism (or informationalism) is becoming a dedicated philosophical process for the development of information philosophy. Informationism is attempting to understand arbitrary phenomena as information-like.

1.5. Informationness and Informationality

Informationness denotes the broadest property, quality, nature, characteristic, and attribute of information concerning a context, phenomena, information, etc. In this manner, one can understand the informationness of a molecule, cell, organism, brain, being or machine stating that all these objects have properties being in some respect information-like.

Similarly, informationality is the property of Informational. So, informationality of a living cell processes is the understanding of metabolic, replication, and immune reactions as information processes. At this point, a living cell processes are understood as informational processes.

1.6. Informationable

In accordance with the philosophy of information, it is possible to understand everything as information or Informational. One

is able to perceive, recognize, and understand everything as information. On the other hand, every form or process is able to inform or to generate information in other forms and processes. So, forms, processes, everything is in the realm of the Informationable. The meaning of the adjective informationable and the noun Informationable is becoming senseful in the domain of Informational.

2

Every phenomenon has the property to inform itself and other phenomena through its Informing.

In this paragraph the archetype will be 'to inform' and the following sequence of derivatives will be considered:

to inform: Informing, informism, informic or informical, informistic or informistical, informness, informingness, informity, informable.

2.1. To Inform

Information has the property of informing. Information has the capability to receive, modify, change, alter, develop, grow, vanish, bear, generate, transmit, form, process or inform its own and strange information. Information informs in all possible and yet impossible ways. The verb inform represents a passive and active phenomenon of information. Also, information informs itself alone and other information as an arising phenomenon, in which information within information is coming into existence. So, 'to inform' also has the meaning of 'information to come into existence' or transparently the meaning of any other imaginable phenomenon.

If the coming of information into existence is the main prerogative of informing, then one must explain, what the broadened meaning of coming into existence is. Stating that information is coming into existence means that new information is arising, that the existing information is modified, altered, or changed, as well as vanishing, that other, outside information is influencing the instantaneous information, that information existing in this time slice is influencing other, outside, future information and itself, that information is coming into existence from the previous information, from itself, and from other information-related or non-related sources, and that information is arising also in a free, unexpected, unforeseeing, unpredictable, and unusual way. So, the meaning of the verb *'inform' has to be understood in a way*, where information comes into existence and this coming into existence is active and passive informing in all possible modes.

In this context of meaning, the Latin verb 'informare' has had colloquially narrowed meanings as to form, shape, instruct, teach, represent, show clearly, describe, reflect, etc. Current meanings of these verbs can be, for instance, to form complex thoughts, to shape functions of the brain, to instruct a living information system, to teach the most complex philosophy, to represent everything of the surrounding world as images of thought in the cortex, to show ideas clearly in the form of language, to describe everything by information, to reflect in human mind, etc. In this sense, the meaning of the verb 'inform' is becoming semantically satisfactory for further investigation of information.

2.2. Informing

The verbal noun 'informing' represents the action in which information is informing. Sometimes this noun will be written with the capital initial letter (Informing) to clearly distinguish it from the inflected verbal form. It becomes quite clear how Informing is informing (= Informing informs), information is informing, information is Informing, etc. Informing is again either active, passive, or both. Through Informing information is coming into existence as described in the previous paragraph. Similar to the noun 'information' and the verb 'inform', Informing has the highest imaginable semantical power of its predecessor. Therefore, Informing is expressing all possible activity of information, when *information is informing on its way to* information and when new information is coming into existence, arising from previously existing information.

A broadened meaning of Informing may be compared to the Greek 'poiesis'. The meaning of 'poiesis' is acting, composing, creating, doing, elaborating, establishing, executing, finishing, forming, getting ready, making, manufacturing, originating, poetry, preparing, proceeding, producing, setting up, working, etc. As with the word information, the word Informing now passes through a corresponding change and enrichment of its meaning. Any phenomenon, its arising, acting, behaving, and processing can be understood as Informing.

Informing is not necessarily dependent on the previous information. Informing brings to the surface also new, unexpected information. At this point, it is necessary to stress how Informing is not an indispensable, causal activity, even though causality is squeezing through in various forms the natural and social phenomena. Because not all phenomena are causally connected, they can occur accidentally, e. g., as an external information entering through sensory ways to cortices or as a circular, cortex internal information (e. g. thrownness and breaking down in Winograd, UCC, 33-37). *Examples of causal and non-causal* Informing are also perturbing (communication among beings) and autopoiesis (biologic self-production) (Maturana, OL). Informing of information and information qua Informing are generating a new regular form or process which will be called *counter-information*.

2.3. Informism, Informness, and Informingness

Informism is nothing less than the thinking and understanding of the entire phenomenology through Informing or by means of Informing, through the way on which information informs everything - living and non-living. In the context of informism the adjectives informic (informical) and informistic (informistical) can be used denoting some properties of this reasoning. Informness and informingness are consequences of this reasoning and represent the properties, how all phenomena are understood to be informing or Informing.

2.4. Informity and Informable

Informity is the property which a phenomenon has in regard to its information, to its Informing, to its Informational, to its form or process which is information-like or can be explained or understood as such. In the

framework of informism, the informity is the most general property which is particular to each phenomenon or phenomenon occurrence.

Every phenomenon is able to inform to a necessary and sufficient extent, complexity, clarity, detail, and depth in itself and outside itself. So, in general, phenomena are informable and there exists and arises the Informable about a phenomenon. Evidently, the Informable is informationally two-folded: it is simultaneously perceiving-sensitive and behaving-active.

3

The informative is bringing information somewhere to somebody. However, the informative is also causing or generating the informing in beings and populations. Hence, the informing of the informative is coming into existence.

In this section the archetype 'informative' and its derivatives will be determined. So, the chain

informative: informativism, informativeness, informativity, informativable

will be considered.

3.1. Informative

Can the adjective 'informative' attain the meaning of the adjective 'informational'? In everyday language informative has the meaning of the imparting knowledge and of the instructive. By general agreement, the meaning of informative could be broadened in the way to attain the meaning of informational. In paragraph 1.3 the difference of meaning between informational and informative is pointed out. So, one can use the adjectival form 'informative' in the ordinary (narrowed) way.

3.2. Informativism

As derivative of informative, the noun informativism concerns the meaning closely connected with the particularly narrowed meaning of informative. Informativism is a non-complete, particular thinking about information. In thinking, the essence of information (Being of Information) is excluded, so, informativism is thinking through giving, processing, carrying, possessing, bringing, mechanically transforming, translating, transmitting, receiving, assembling of information, etc. In this context, informativism is merely a part of informationalism.

3.3. Informativeness and Informativity

Informativeness and informativity are close to the property of 'being informative' and 'informativism'. If the first one sounds more expressive, the second one is more sensitive in respect to an informative subject or informative object.

3.4. Informativable

The informativable is a form or process, which expresses the ability to be informative, to carry, give or bring information. The emphasis of information in an informativable case is in propagation of information in an informationally passive way. Hence, the informativable is semantically included in the informationable.

4

In this section meanings of the following words will be presented:

informatics: informatical, informatism, informaticness, informaticity, informaticable.

4.1. Informatics

Informatics is a young, arising discipline which concerns many particular fields, where information is important for the development of these fields. In this way, informatics has many specializations and field orientations. The term 'informatics' developed in the European university community as the counterpart to the term 'computer science' used at universities in the United States. Afterwards, the meaning of the term was broadened. Today, informatics concerns information and information investigation in several fields and disciplines as a general means, as philosophy, as education, as science, as technology, as application in economics, engineering, medicine, biology, sociology, and in other related disciplines. Informatics is becoming the way of thinking, methodology, knowledge, and processing, all of which concern information, informational forms, and informational processes in a general sense. Informatics unites the different informationally developing disciplines dealing with theory, philosophy, technology, methodology, terminology, and application of information.

Described as such, informatics in the realm of information may have similar role, for instance, as mathematics has in the realm of mathematical and semi-mathematical objects in different sciences and disciplines. So informatics, with its characteristically shaped methodology, tools and objects, may have a general developmental value for several other disciplines. Informatics is dealing with information systems and how these systems in a man-machine interaction exist as living and technologically supported information systems. An information system is no longer a computer system isolated from man and his life processes. So far as computers of the future will become information machines, computer science will be replaced more and more by information science.

4.2. Informatical

The term 'informatical' is used as the adjectival form regarding informatics. Informatical means are information objects, methodologies, philosophies, theories, technologies, tools, etc. The informatical includes the entire realm of informatics, the field of informatical means. There are essential, meaningful differences among the usage of the adjectives informational, informative, and informatical. These differences are now already surfacing into one's awareness. The same is valid for the corresponding substantival forms of the Informational, of the Informative, and of the Informatical.

4.3. Informatism

Informatism is the way of understanding, how, why, where, and when informatics, as a discipline, methodology and tool, can be used and has to be applied. Informatism concerns thinking, questioning, problem solving, and different implementations in different fields of everyday life by informatical means. Informatism is becoming the main stream of orientation in modern life, an orientation for surviving in a man-to-man's, biologic, and cosmic hostile environment by informatical means.

4.4. Informaticness and Informaticity

Informaticness and informaticity are closely related terms and the usage of the first or of the second one will depend on one's linguistic taste. Both refer to informatics and informatical, so, their meaning is informatically transparent. Informaticness or informaticity is becoming the imperative of usage and of application of informatics in everyday life.

4.5. Informaticable

To be informaticable means to have the possibility (of possessing the property) to be informatically expressed, perceived, or used. In general, every form or process, which is informaticable, can be governed or presented by informatical means, by methods, tools and approaches of informatics. This fundamental property calls for usage of informatical means in every field of human activity, in every technological discipline, in life, and in non-life processes.

5

The chain with the archetype 'to informatize' will be the following:

informatize: informatizing, informatizism,
informatizeness,
informatizability, informatizable.

5.1. To Informatize

The verb 'to informatize' has the typical meaning of 'to inform actively', 'to influence by informing', 'to change by means of information', 'to act through informing', 'to come into existence', 'to arise', 'to become', 'to evolve', 'to generate through the influence of information', etc. Information is informatized by itself alone, i. e., it is informing actively, being influenced, being changed, coming into existence, arising, becoming, evolving, generating, etc. Information informatizes through changing, arising, generating itself and other information. If sometimes 'to inform' sounds unclear regarding the activity of information, 'to informatize' always has an active and clear meaning. If 'to inform' has the meaning of sending information without an informational consequence, 'to informatize' always means an active influence onto the informational receiver. In this respect, 'to informatize' has a sub-meaning regarding 'to inform'.

5.2. Informatizing

Informatizing is the verbal substantive of 'informatize' concerning the meaning of its verb. One says that information is informatizing only in such cases when information is influencing itself and other information in a generative or modifying way. Informatizing has the meaning of informational activity or of an active change of information by information.

5.3. Informatizism

Informatizism represents the active position of information through informing of information. In contrast to informativism, which may be a passive receiving of information, informatizism understands informing in its compulsory changing or informationally generating manner. Informatizism is a changing or generating understanding of information and informing by itself. In this respect, informatizism is, for instance, characteristically non-dogmatic, non-blinded, always on the way to its own breakdowns, which influence and change its understanding. Informatizism is continuously and actively changing itself.

5.4. Informatizeness and Informatizability

Informatizeness and informatizability are akin notions with slightly different meaning. They carry the meaning of the verb 'informatize' and the meanings of its derivatives, and express the possibility of information or of any phenomenon to be informatized, etc.

5.5. Informatizable

It is to say that not all information is unconditionally informatizable. An information is informatizably resistant, if there does not exist a way to change it by information. On the contrary, information which can be changed through informing or informatizing, is informatizable. Informatizable information has the property to be changed, to come to existence, and to be generated from other information or from itself. Information, which is informatizable, has the property of sensibility to other information and to itself. This sensibility is the ground, cause, or reason for alteration and generation of informatizable information.

A phenomenon is informatizable, if it can be expressed informationally, i. e. if its form or its process can be represented informationally. In fact, every perceived phenomenon is informatizable, otherwise it could be not represented on the sensory or cortical level. Hence, non-informatizable phenomena are not perceived informatically and their informational status is as they do not exist. The fundamental hypothesis of information is that every phenomenon is informatizable and that it can be treated as an informational form or an informational process. To be informatizable means to be in the realm of information.

6

The last semantic sequence which will be considered is

informatization: informatizational,
informatizationism,
informatizationess,
informatizationable.

6.1. Informatization and Informatizational

Informatization denotes a form or a process of informatizing or an active form or process of informing. This kind of forming or processing is denoted as informatizational. Similarly, as mathematization or automatization concerns mathematical or automatic means, informatization concerns some transformation from phenomenological as well as informatical into informatical. Informatization is a process of making things informative, informational, or informatical in an informatizational way.

6.2. Informatizationism

Informatizationism is the understanding and belief that information can be obtained or used in any governing of phenomena and that various ways of informatization are senseful, rational, economic, and necessary for survival. In this way, informatizationism can influence the motivation of discovering new informationally adequate substances and processes, for instance, tactilizing and other molecular and biological processors, which may be applied in future informational or intelligent machines (LMC).

6.3. Informatizationess and Informatizationable

Informatizationess expresses the possibility that a form, process, or phenomenon can be informatized, can accept some informational forms and processes, and can informatize in a complex way. In a similar regard, forms, processes, or phenomena are informatizationable if it is possible, by informatization, to enable them to function informatically or to observe informationally evident events in them.

7

7.1. Informational Forms and Informational Processes

Informational form concerns a particular form of information. If information has the meaning of being-in-the-form or being-in-the-process, then informational form has the meaning of being-in-the-form-as-a-form or being-in-the-process-as-a-form. Similarly, informational process takes the meaning of being-in-the-form-as-a-process or being-in-the-process-as-a-process. This cross-meaning of form and process presents how information can exist as form of a form, as process of a process, as form of a process, and as process of a form.

Generally, form of a form is not a trivial situation, because information is investigating itself as a form, where investigating and form are becoming counter-informational. The similar holds true for process of a process, where an informational process is investigating a process as information, generating information of advancing, developing, or arising of a process, etc. It seems evident, that informational form and informational process are informationally regular and therefore in the realm of information.

7.2. Information and Counter-Information

Information is on the continuous way to information as form, as process, or as both. On its way, information is coming into existence and arising through its counter-information. What does counter-information represent in an informational context? Information is coming into existence anywhere in a substance as a process within information. This process itself can be understood as information and in the continuous processing also as counter-information. So, the following appears: information is gaining counter-information, which is becoming information etc. This process is also information gaining counter-information etc. Generally, information and its processing is information.

On the conscious level in a being's brain counter-information can be understood as the consequence of questioning in a problem phenomenon. In this case, the initial question raises questioning which produces some kind of interrogated. The interrogated is the appearance of counter-information. But the interrogated has its return-influence on questioning, so that questioning is becoming counter-informational. In this way, counter-information, which sprouted first in the interrogated, is transferred backward to questioning, and from questioning even backward to the question itself, gaining, amplifying, or generating counter-information. In this way, counter-information is arising in a mutually forward and backward dependent process of question, questioning, and interrogated.

Examples of arising of counter-information can be found anywhere in the living and non-living world. But this does not mean that arising of counter-information exists in contemporary technological machines. Even though counter-information is a regular informational phenomenon in a living being, it does not appear in artificial mechanisms, methodologies, algorithms, computer programs, and machines made by man. This statement is extremely important for a designer, who is trying to overcome a particular informational problem called intelligence, where intelligence appears as a kind of counter-information.

It seems that counter-information is arising as a consequence of informational investigation within information. This investigation principle generates new information and causes the arising and coming of information into existence. Even this informational investigation principle alone is information, which depends on blindness, illusion, or breakdowns which are embedded as informational phenomena within the arising information. In this way, every counter-information, regardless of where and on which informational level it arises, is characteristically blinded, illusive, or broken-down and underlies a characteristic form of a system's strategy or a being's metaphysics.

7.3. Informational Recurrence

Informational recurrence is a term in which the meaning of recurrence is much more powerful than its usual meaning. Contrary to common sense, in addition to the property of being recurrent, informational recurrence has the

property to be informationally generative and informationally parallel. In this context, the generative is concerned with arising or coming of information into existence. Informational arising can be understood as a particular form of recurrence, where the arising of information is recurrent. The informationally parallel concerns the arising of parallelism in form and in process within information. Hence, informational recurrence is regular, generative, and parallel recurrence in the described sense.

If information is being-in-the-form or being-in-the-process, and if recurrence is being-recurrent, being-generative, or being-parallel, then informational recurrence can be expressed in a number of ways, combining all possible alternatives. A complex example would be the following: being-in-the-form-and-in-the-process-as-regular-generative-and-parallel-recurrent.

7.4. Forms and Processes of Life

Information is coming to its full expression as a form or a process of life. Where does the life begin? If life is based on the appearance of molecules of life, these molecules already concern information which is present in the form of genes. Genes seem to store static information which governs the evolution of a being during its development in a hostile environment as well as a being's main life forms and main life processes. But information which is represented in genes can suddenly change in some of its details. This occurrence of change is called gene mutation. So, generally, genes are informatically (chemically) not absolutely stable and represent information, which can be altered under unforeseeable circumstances. The consequence of this fact is the evolution of species.

Molecules of life (DNA, RNA, proteins, and dedicated proteins called enzymes) demonstrate different forms and processes of information. If the deoxyribonucleic acid (DNA) proves an outward form of a temporarily stable information, the ribonucleic acid (RNA) is carrying different and particular information. In this molecular mechanism the enzyme represents a real biologic information machine reading information stored in an RNA and on this basis synthesizing the corresponding protein from lumps of the biologic environment.

The next higher form of life is the living cell. The cell is a biologic machine, an organism using molecules of life and other molecules for various informational manipulations. On the cellular basis, the immune system is an extremely sophisticated information system. The aim of this system is to recognize pathologic invaders by means of molecular receptors and neutralize them by means of a special biologic mechanism. In neurons, which are nerve cells in the brain, various receptor and transmitter substances are used for information transmission and information arising. A major function of the neuron's cell body is the synthesis of macromolecules. From modern cell biology, it is known that information for the synthesis of macromolecules is encoded in the DNA of the chromosomes within the cell's nucleus. In all cell types, there are two basic ways in which this information is processed. The genetic information is passed from parent to daughter cell during cell division (heredity) or a selected portion of the genetic information is transcribed into RNA and translated into proteins (gene translation).

Like most cells, neurons contain the complex apparatus for synthesizing proteins and the genetic information in DNA to encode them. They also have mitodrondria and enzymes for biosynthesis and for metabolism. Nerve cells are excitable and specialized, so they contain one or another transmitter substance, special ion channel, membrane transport mechanism, or type of receptor molecules. The understanding of an information process in a neuron ultimately depends on the identification and characterization of all these components. Thus, the neuron is a complex information machine being informatically specialized.

7.5. Intelligence

Intelligence is specifically structured, organized, and oriented information. Intelligence concerns different forms and processes of man's life, of man's conscious activity, and of man's culture. Intelligence is an informationally complex structure, problematically connected with a particularly oriented information, informing this information in a particularly resulting way, and producing the so-called instantaneous solutions of the given problem. Intelligence is a particular information concerning a particular problem domain and giving informational results in a form of particular informational forms and particular informational processes. In this respect, every intelligence is concerned with its own blindness and its own breakdowns. Intelligence's blindness originates from its particularity, peculiarity, and orientation in regard to the entire, broader information realm. Intelligence's breakdowns interrupt its blindness and enables its development on the basis of new, intelligence-relevant information which arose during the intelligent investigation within intelligence itself.

Intelligence is an informational process of appropriation (or seizure) of its structure, organization, and orientation. Intelligence is informing appropriately, properly, and aptly regarding the problem domain. This domain occurs in intelligence as an object information being informed by itself and by intelligence. So, a problem information is coming into existence and depends solely on the depth of the problem solving intelligence, which is the information of the problem solving degree, depth, structure, organization, and orientation. This appropriation of intelligence is causing intelligent blindness, but also intelligent breakdowns. In this regard, every intelligence is intelligently relative, possessing its characteristic blindness (intelligent illusion or intelligent stupidity) and informing its breakdowns (intelligently essential facts coming to the surface only in particular cases).

As information, intelligence is always embedded in an broader informational context. Intelligence is coming into existence in an information-related system, so the structure and organization of information in this system influences its shaping and design. Intelligence is circularly concerning intelligence and other information which is intelligence-related and system-dependent. Intelligence is developing itself alone, rising itself to higher and different informational levels. At this point, it is obvious how information complexity, arising, and developing on the one side, and intelligence coming into existence on the other side, are very closely related processes. Thus, intelligence cannot be quite clearly distinguished from the general and the

broadened informational background. It seems that intelligence is only one of the particular forms of information, which arises out of the informational background. Within this background, intelligence has its own structure, organization, and developmental orientation, which make it 'intelligent'.

Informationally, intelligence always concerns a particular field, problem, or situation. In this respect, it is possible to speak about intelligence, which is more or less specifically oriented, which concerns less general and more specific matters, activities, and forms. As in information, a similar general principle of intelligence can be observed, which is arising from investigation, questioning, and counter-Informing. Obviously, it has to be said that intelligence is an informational form with characteristically problem-oriented and powerful, counter-informational investigation, where a particular solution for a specific problem situation is being sought. Intelligence is similar to a questioning system, where problematic questioning is its main informational process and the problem-interrogated is the main instantaneous result. It is becoming clear how intelligence is a circular informational mechanism concerning the investigation circle of question, questioning, and interrogated. This circular process is a living, arising, and developing form, which changes its structure, organization, and orientation, performing dynamically on the intelligence itself.

Intelligence is a natural informational property and hitherto, a technological form of intelligence has not been implemented. In primitive and complex living systems this property does always exist. The purpose of intelligence in the living world is the surviving of species. Intelligence on the level of consciousness can dwell outside of the surviving principles as a form of cultural activity. From this viewpoint, artificial intelligence, as of yet, does not exist.

Intelligence falls into the abyss of information. A being's information falls into the abyss of a being's neurophysiology. Intelligence does not really see the bottom of its abyss and as it is thrown into depth, it falls upward to the top. Although the bottom of intelligence's abyss is informational, the way from the top to the bottom is not seeable yet. This route of coherence between the abyss' top and abyss' bottom is blinded and intelligently (informatically) hidden.

7.6. Cultural Forms

In the realm of man's culture many various forms exist, which can be investigated from the informational point of view. For instance, some of these forms can be grouped in the following way:

creativity, language, and speech;
philosophy, ideology, science, and technology;
aesthetics, rhythm, harmony, art, and literature;
crisis, capability, and incapability; and
functions of the mind, information systems, and artificial intelligence.

These cultural forms are essentially influenced by the conscious way of man's thinking and behaving. In the following sections some of these forms will be informationally analyzed in a more detailed way.

8

" ... man does not know himself as the one who is being brought into relation to Being; that is, he does not know himself as man. Ruled in this way, man today, despite what seems true to him, never encounters himself, i. e., his essence."

(W. Lovitt in Heidegger, QCT, xxxiii)

8.1. A Being's Information

Beings are informing as long as they live. How is a being's information formed and processed, how is it structured and organized? From the informatical point of view, a being is its total information, which is highly composed and connected. Limitations of a being's total information appear on the level of a being's informational autopoiesis, which represents an autonomous, self-productive information system. In a biologically, highly developed being information is arising in forms and processes from molecules of life toward the particularly structured and organized cortices. Informational structure and informational organization are impeded by an informational hierarchy, which is the consequence of substantial structure and substantial organization. The informational hierarchy is characteristically structured and organized autopoiesis of a given species on the first (genetic) level and of a concrete individual on the second (metaphysical) level. The biologic substance is autopoietically conditioning the nature of the information structure and information organization up to the metaphysical level, and of informational forms and informational processes governing the life, behavior, and imagination of a being.

For a being, only its own information is relevant. As long as outward information is not perceived by the being, this outward information is not coming into the context of a being's total information and therefore does not receive any informational relevance. This kind of outward information represents an irrelevant informational noise. A population of beings is characterized mainly by the population-relevant information. This population is informing (generating and accepting) its own and characteristic system information, which governs the information system of a given population. Furthermore, from the informatical point of understanding, a population of beings can be comprehended as a more composite and more complex being (the Being of Population) than an individual being is. Similarly, a being or an individual is a complex of cell populations, a cell is a complex of macromolecules, a macromolecule is a complex of molecules, etc.

A being's information constitutes all possible and arising understanding, cognition, behavior, and relevance of a being's life. The information hierarchy of a being produces the arising of intelligence and other higher informational functions. The possible information coming into existence is a consequence of informational, substantial structure and organization and of informational and substantial processes. Physiological structures of a being are consequentially connected with the biologic variation, selection, and reproduction governing the life and the behavior. On the behavioral level, a being is always solving life problems, and within this solving, information is arising as an evaluation

principle, which recognizes how problems are solved. In this processing, the evolutionary learning, variation, selection, and reproduction are essential information processes.

Informationally, a living being constitutes a circle, in which physiological structures are preoccupied by evolutionary learning, variation, selection and reproduction, delivering and developing information for a being's life and behavior. Sensory information enters into a being in the form of various signal patterns, which are accepted by adequate physiological structures. These structures are themselves under the structural influence of a being's information. A being's information is circular concerning the biologic structure and biologic organization. However, all these processes can be understood as information processes, where information about biologic structures and organization always exist.

8.2. Being and Time

A being, beings (Being), metaphysics, ontology, appropriation, activity, time, etc. are all information in the realm of everyday phenomenology and remain in the realm of information. The realm of information is also information by itself. Both, Being and time, are regular information, which are regularly interwoven by information.

Being can be understood as all possible and also as non-possible information concerning beings as beings. In philosophical terminology, Being never changes. This may be a consequence of the particular philosophical doctrine called ontology. This philosophical view generalizes the grasping of Being, where Being is observed as an absolute entity standing comprehensively outside of beings themselves. In informational terminology, Being as information is always coming into existence by the informational arising of Being. Thus, Being is merely a particular information concerning beings and regularly changing information. Similarly, this holds true for philosophy, which is an informational process informing Being as an informational object. From the philosophical point of view, there is a contradiction between time and Being, which was pointed out by Heidegger (OTB, 3-4): "Philosophy knows a way out of such situations. One allows the contradiction to stand, even sharpens them and tries to bring together in comprehensive unity what contradicts itself and thus falls apart. This procedure is called dialectic." From the informatical point of view, contradiction, dialectic, etc. is merely particular information, if they are arising as information or in the informational realm at all. A contradiction in the informational treatise of Being and time does not appear informatically.

Evidently, Being has to be understood as information which is independent from beings and surpasses any being's information. The Being is information concerning all beings and being composed in this way from information coming from individual beings. Thus, the Being as information is nothing else than beings-in-the-form and beings-in-the-process.

Information is circular concerning any information as well as time. What is circularity of information as information? Information as a subject has its object - information. Time is the relation between information as subject and information as object. The consequence of this relation is the arising of information where subject is coming into existence from object and from itself alone, and also when object is becoming object from the subject. This internal information mechanism of circularity or of informational subject and informational object, is time, the informationally inward understanding of time. Time as information is circularity of information as information.

Circularity of information is information recurrence. This recurrence is circularly generative and circularly parallel in a spatial and temporary sense. Thus, information is coming into existence in a circular, generative, and parallel way. Time as information is always informationally recurrent in the same way as arising is coming into existence, from past to present or from present to future. Time is information dealing with circularity, iteration, recurrence, generation, parallelism, and with other informationally arising and peculiar forms and processes.

Since a process of information is information, time concerns processing of information and thus, is a regular informational phenomenon, a regular informing of information.

8.3. Phenomenology

Information is informing and through informing represents phenomena in a general and in an arbitrarily detailed way. Phenomenology is a particular information regarding various phenomena and investigations of phenomena. In its broadest and most fundamental meaning, phenomenology (from the Greek 'fainomenon' and 'logos') is the science concerning phenomena. Its initial preoccupation was to discover the difference between illusion and truth. However, as phenomena appear on the conscious level of the mind, it is also becoming a science which describes, and explains phenomena of consciousness. Phenomenology can also be understood as a general theory of empirical phenomena, in contrast to a science, which handles things by themselves alone. A metaphysical meaning of phenomenology was attributed by Hegel's analysis of forms and matter as phenomena of mind. "The term 'phenomenal' (the living individual, his behavior) is subordinate to the term 'generative,' which appears as an anonymous program - produced, so it would seem, by the most anonymous of cosmic actors: chance." (E. Morin: Self and Autos, in AP, 128.)

Today's phenomenology is a particular field of philosophy, whose originator, E. Husserl, refused subjectively abstract cognitive-theoretical trials and has posted critical foundations for scientific thinking by his 'phenomenological method' in his thesis 'back to things'. As a universal philosophical and as a general scientific fundamental discipline, phenomenology has essentially influenced modern philosophical thinking against the naive realism, positivism, rationalism, psychologism, constructivism, and formalism. In this respect, phenomenology can find its appropriation also in informatics, information theory, information philosophy, and information technology.

8.4. Metaphysics and Ontology

Metaphysics will be the label for the entire, total information of a being. Ontology will be the label for the entire, total information of a population. Since society can be understood as a population of populations, the entire social information can be labeled as sociology.

A being's information is metaphysical. A population's information is ontological. Whereas metaphysics as information is individual, subjective, and autopoietic, ontology as information is intersubjective, outside, and cultural (poietic) information. Through culture, ontology is being more or less accepted by a being, and in this way, this acceptance is becoming metaphysical. How is information becoming metaphysical and how is it becoming ontological?

To some extent, metaphysics, as a being's information, autopoietically concerns a being. H. R. Maturana (Autopoiesis, AP, 21) gives the following dictum: "A system is autonomous if the relations that characterize it as a unity involve only the system itself, and not other systems. Thus defined, autonomy can be viewed as a central characteristic of living systems. Yet, since autonomy is not necessarily a feature exclusive to living systems, any attempt to explain the organization of living systems must show how they are autonomous and how all the phenomena proper to them arise as a result of their autonomy. It is in this context that I maintain that the notion of autopoiesis fully characterizes living systems as autonomous entities in physical space." In this respect, metaphysics is autonomous, where autonomy is comparable to a being's subjectivity. Although ontology can be viewed as an autopoiesis of the being's population, it can also be viewed as autonomous in some broader, population-characteristic aspect. Hence, ontology is population-subjective.

Philosophy is searching for an absolute or objective grasping of ontology, which is not dependent on metaphysical and culturally ontological phenomenology. If population is population-autopoietic, then the questioning for an objective ontology remains open. Thus, the interrogated always appears as a new form or a new process of autopoiesis. Hence, Being appears to be understood only as a form of culturally autopoietic grasping, where culture is developing Being from one autopoietic form into another. Informationally, it would be possible to understand how cultural metaphysics (or ontology) is proceeding from the present to a new form, which similarly holds true for a being's metaphysics.

In the information hierarchy of metaphysics, ontology and sociology, the mutual influence of all these informational forms is coming into existence. Pure autopoietic information does not really exist; simultaneously, this information is allopoietic and heteropoietic. In this concern, metaphysics of different beings influences population's ontology and society's sociology to an informational extent and vice versa. This is similarly true for ontology and sociology.

9

9.1. Autopoiesis

The Greek 'autopoiesis', as originally conceived by H. R. Maturana and F. J. Varela, means literally 'self-production'. Autopoiesis is a label for a particular meaning of self-production. Self-production phenomena are observed informatically in living systems. The living cell is an extremely complex self-production system, synthesizing DNA, RNA, proteins, lipids, enzymes, etc. and consisting of a hundred of thousands macromolecules. Throughout this changing of life forms and life processes, the cell demonstrates its distinctiveness, cohesiveness, and autonomy. The living cell is producing itself. The living cell is autopoietic or self-producing in a particular way.

Autopoiesis can be determined by specific forms and processes, such as unity, production, organization, structure, autonomy, system, topology, etc. Processes in an autopoietic system are organized circularly, as they depend recursively on each other; they form a recursive closure. Production processes are synthesizing, transforming, or destroying various components which appear in an autopoietic system. For instance, destruction of components produces substrate on which processes of production can act. A unity assembles its components. The autonomy of a unity depends on information which is the function (the Informing) of a unity and the cognitive preference of the observer. Some parts of a system's structure allow the observer to identify them as unities.

By its nature of coming into existence, information, as a form and as a process, is also autopoietic. The main characteristics of autopoiesis also hold true for information. However, information is not only autopoietic; it can generate information which stands outside any autopoietic system. By definition, an autopoietic system has a closed organization of production processes, where this organization of processes is generated through interaction of their own products and a topological boundary emerges as a result of the same constitutive processes (M. Zeleny, AP, 6).

Autopoiesis, as cognition of an autonomous living being's behavior, is a characteristic informational form, which came into existence as a specific result and construct of a being's information. This informational discourse is a consequence of a self-referential understanding of beings' behavior. Also within autopoiesis as information, information is arising through information - by other information and by itself - to keep the thread of causality but also to stay in the position of non-causality or of the spontaneous coming of processes into existence.

Autopoiesis as a cognitive discipline brings the following, informationally essential questioning to the surface: If a living being is autopoietic, does its information arise in an autopoietic way? Does this mean that a being's information is particularly limited by its own autopoiesis (structure and organization) and that it cannot be developed or come into existence outside of its own autopoietic limits? In the case of a being, it also seems that the autonomy of an autopoietic system is influenced by a being's outward information (allopoietic or heteropoietic information), which might not be consistent with an autopoietic information. Autonomy, as an informational form, is also varying and arising and therefore, is not a rigid or constant informational form. In this regard, autonomy as a constituent of an autopoietic system is varying the system's autonomy and also the system's closure. At this point, one can understand that strictly closed systems do not exist: they are only temporarily, conditionally, or informatically closed in a particular information realm.

9.2. Neural Science and Information

The relation between neural science and the mentation occurring in neural systems as informational forms and informational processes

is the key point of modern information theory. Mind is the integral information process of a being's brain. Forms and processes of the brain can be understood informatively, informationally, or informatically as structurally and organizationally connected biologic, physiologic, physical, chemical, etc. Informational reflection and informational imagination unite all these different, informationally consistent and non-consistent forms and processes of the neural system.

Neural science concerns specific neural forms and neural processes in a general and detailed manner where these forms and processes can be ordinarily comprehend as information, Informing, Informational, etc. What in fact is neural science and how does it deal with many different forms and processes, which seem to be information-concerned? According to the principles of neural science (Kandel and Schwartz, PNS), this science incorporates the following key disciplines:

brain, nerve cells and behavior;
cell and molecular biology of the neuron;
elementary interactions between neurons: synaptic transmission;
functional anatomy of the central nervous system;
sensory systems of the brain: sensation and perception;
motor systems of the brain: reflex and voluntary control of movement;
the brain stem and reticular core: integration of sensory and motor systems;
hypothalamus, limbic system, and cerebral cortex: homeostasis and aruosal;
localization of higher functions and the disorders of language, thought, and affect;
development, critical periods, and the emergence of behavior;
genes, environmental experience, and the mechanism of behavior;
brain fluids and their disorders;
neurophthalmology; and
the flow of ionic and capacitive current in nerve cell.

In all these disciplines, neural science is deeply and essentially concerned with various information systems. It brings into the foreground informational concepts, e. g., biological, physiological, chemical, physical, etc., which are subsystems of the integrated neural system.

Since life is a being's behavior and behavior informationally governs a being, the particular question which arises is, how can the Informational be comprehended in genes, in environmental experience, and in the mechanisms of behavior? Cognitive psychology and psychoanalitic theory recognize that the diversity and complexity of human experience is based on genetic and learned factors in determining the mental world representation, and they postulate that behavior is based on this representational information. According to I. Kupferman and E. R. Kandel (PNS, xxix-xxx) the following key subjects can be treated:

genetic determinants of behavior;
learning; and
cellular mechanisms of learning and the biological basis of individuality.

9.3. Brain and Behavior

But all the time the brain is still,
a man can think properly.

(Hippocrates, Fifth Century, B. C.)

Brain is a living substance and behavior is a living processing. Brain is the commanding apparatus of a being's behavior. Brain is a sort of information machine, which enables and handles information processes of behavior. The brain also influences and in some way structurally and organizationally limits behavior, and behavior has similar developmental and generative influence on the brain. Behavior is a reflection of elaborate brain functions and these functions (informational processes) frame the so-called mind. Brain and behavior are particular examples of information forms and information processes.

Brain consists of basic informational units which are the neurons and the glia cells. These basic units govern the most elaborate cognitive functions (behavioral processes). The functioning of neurons in the brain is influenced by the behavior, which comes as a consequence of sensory information from other beings as well as from other environmental phenomena. However, groups of neurons are structured (functionally connected) within the nervous system into regional subsystems (regions, areas, nuclei), so that behavioral processes can be localized in specific areas of the brain. Regions of the brain are specialized for many different processes, which constitute particular behavioral information - mental and motorial. The central nervous system is bilateral and essentially symmetrical information machinery, which consists of the following main parts:

1. The spinal cord receives information from the skin, joints, and muscles of the trunk and limbs, and sends out motor information for the reflex movement and voluntary movement.

2. The medulla oblongata is the supreme extension of the spinal cord and incorporates, in addition to other nuclei, reflex nuclei for breathing, functioning of heart, coughing, vomiting, etc.

3. The pons and the cerebellum are supreme to the medulla. The cerebellum is modulating the force and range of movement.

4. The midbrain is supreme to the pons and lies between hindbrain (the medulla, pons, and cerebellum) and the forebrain (the diencephalon and cerebral cortex).

5. The diencephalon contains two informational relay structures. The thalamus processes most of the information coming from the rest of the central nervous system to the cerebral cortex. The hypothalamus controls autonomic, endocrine, and visceral integration.

6. The cerebral hemispheres consist of the basal ganglia and the overlaying cerebral cortex. Both the cortex and the basal ganglia are concerned with higher perceptual, cognitive, and motor information.

The medulla, pons, and midbrain constitute the brain stem which contains the cranial nerve nuclei which in turn receive information from

the skin and muscles of the head and from the special senses of hearing, balance, and taste. Other nuclei regulate motor output information to the muscles of the face, neck, and eyes. The brain stem incorporates another structure called the reticular formation, which determines levels of arousal and awareness.

Another structural and organizational principle of the brain is parallel (information) processing. Brain processes are served by more than one pathway. Within the brain,the cortex is divided into two hemispheres with four anatomically distinct and functionally specialized lobes: the frontal, pariental, occipital, and temporal. In these lobes, planing and movement, somatic sensation, vision, and audition as well as learning, memory and emotion are processed respectively. Each hemisphere of the cerebral cortex is concerned with sensory and motor processes of the contralateral side of the body. The hemispheres are not symmetrical in structure, nor are they equivalent in function (information processing). For example, the cognitive aspects of language processing are localized in Wernicke's area and in Broca's area in the left hemisphere, however, affective components of language as the musical intonation of speech, emotional gesturing, prosodic comprehension, and comprehension of emotional gesturing are processed in the right hemisphere, where their anatomical organization mirrors that which is for cognitive language in the left hemisphere.

Even though, the brain is immensely complex and the structure and the function of many of its parts are still poorly understood, it is becoming more and more evident that the brain is an informatically structured and informatically organized apparatus with specific processes occurring in its parts. These processes - the mind - is an informational function which is becoming more and more understandable in modern neural science and in informatics.

A

Every translation from one to another language is faced with its own way left largely free to the translator. This essay was written originally in English and represents author's insufficient habits acquired in his irregular experience with English. So, let imagination arduously search for missing words in vocabularies and for language's non-dwelled associations in the author's mind. I have to thank John D. Freyder again for his meticulous reviewing and correcting of my "translation" and for hours of intense conversation in which many secrets of the English idiom were revealed to me. Anyhow, this collaboration was a fruitful and pleasant interweaving of philosophical quests and of language's adequacy and language's requests.

B

References which are marked with brackets are used in the text of this essay. Non-bracketed references are recommended for further reading in order to enable a better understanding of the essay.

(LMC) M. Conrad: The Lure of Molecular Computing. IEEE Spectrum. October 1986, 55-60.

J. Haugeland, Ed.: Mind Design: Philosophy, Psychology, Artificial Intelligence. The MIT Press. Cambridge, Mass, Third Printing, 1985.

(BW) M. Heidegger: Basic Writings. Harper & Row. New York, 1977.

M. Heidegger: On the Way to Language. Harper & Row. New York, 1982.

(OTB) M. Heidegger: On Time and Being. Harper & Row. New York, 1972.

(QCT) M. Heidegger: The Question Concerning Technology and Other Essays. Harper & Row. New York, 1977.

M. Heidegger: What is a Thing? Regnery/ Gateway. South Bend, Ind, 1967.

(WCT) M. Heidegger: What is Called Thinking? Harper & Row. New York, 1968.

(PNS) E. R. Kandel, J. H. Schwartz: Principles of Neural Science. Elsevier. New York, 1985.

(OL) H. R. Maturana: The Organization of the Living: A Theory of the Living Organization. Int. J. Man-Machine Studies 7 (1975), 313-332.

(AGB) O. Rieppel: Auf Grenzpfaden der Biologie. Birkhaeuser Verlag. Basel, 1984.

(UCC) T. Winograd, F. Flores: Understanding Computers and Cognition: A New Foundation for Design. Ablex Publ Corp. NJ, 1986.

(AP) M. Zeleny, Ed.: Autopoiesis: A Theory of Living Organization. North Holland. New York, 1981.

(OWI) A. P. Zeleznikar: On the Way to Information. Informatica 11 (1987), No. 1, 4-18.

(Will be continued)

A RELATIONAL DATABASE FOR REPRESENTATION OF MACHINING PARTS

INFORMATICA 2/87

Petar Kočović
SOUR »21. maj«, RO Fabrika Turbo Motora i Transmisije
Grupa za CAD/CAM
Oslobodjenja 1
11090 Beograd

UDK 519.713

ABSTRACT

The paper describes a research effort to directly interface a CAD database with some other systems (CAM , Expert systems , FEM). The research utilize a symbolic representation of Geometry (combined with Topology) , with other data regarding a typical machined component . The topological concept can support an rational database model .

An integrated solid modeling system for representing and manipulating polyhedral objects bounded by bicubic parametric surfaces is presented . Its basic capabilities include a representation of solids , and extended can introduce parametric surfaces and offset tecnique .

Keywords :
Boundary representation , CAD , Constructive Solid Geometry (CSG) , Euler operators Offsetting , Solid Modeling , Surface modeling , Winged-edge data structure

1 INTRODUCTION

Relational database system for representation of solid objects is vital link in the integration of design and manufacturing . Generality of representation of solid objects using relations between geometrical entities and automation in man-machine level focused a research and develop a database . This database is link with Expert systems /MILA82 , MILA84,MILA85, PHIL85/ , because a 3D CAD database is used to represent a 3D object in numerical form .

To input to the type technology sequence , with regard to the machine tool , is as follows /MILA84/:

$$d = (\text{dimension } (d_d) , \text{ kinematics } (d_k) , \text{ operating conditions/energy } (d_{re}) , \text{ technology } (d_e)) \quad (1)$$

The input to the type technology sequence from the branch of technology form is given in this form :

$$a_2 = (\text{elementary form , type of machining , relation between elementary form and type of machining sequence}) \quad (2)$$

To define the type technology sequence , we have the following basic operations as input :

$$a_2^{(o)} = (\text{decomposition , transformations , coupling , sequence }) \quad (3)$$

Some fundamental characteristic are discused below . (For further discussion also see MILA84 and MILA85).

2 TECHNOLOGICAL ASPECTS OF DECOMPOSITION

A technological form is partly shown in visible elementary forms , while those which could appear or are omitted in the transforms , while those which could appear or are omitted in transformation process remain latent . The decomposition process is performed on the visible entities of type forms and higher levels of the workpiece structure . The latent entities do not show in the decomposition stage .

Fig 1 gives examples referring to milling and drilling operations . It has show transformation from initial to final form .

As it shown in Fig 1 all complex which is defined technologically . In very large scale cases all real bodies in mechanical engineering world are build of real geometrical bodies such as cube , cone ,cylindar and sphere. This bodies we call primitive , and these are first class of parts in solid body modelling.

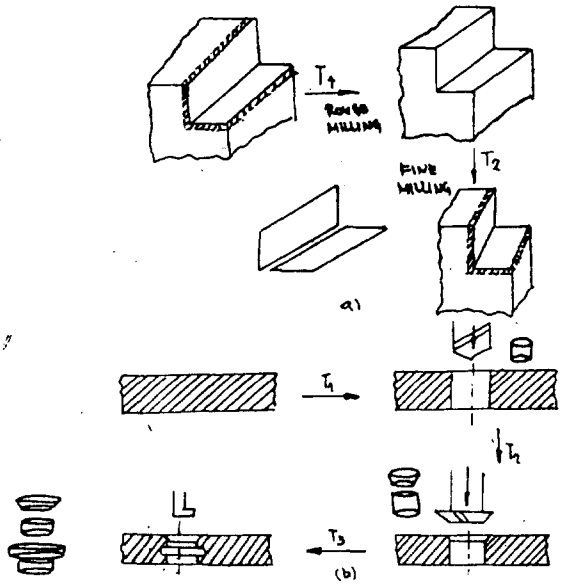


FIG 1 Examples of real solid bodies (primitives are shown in exploded model)

Second class are sculptured (free form) surfaces .

3 GEOMETRIC SPECIFICATION OF PARTS AND ASSEMBLIES

As it shown in VOEL77 , REQU83 , REQU80b , REQU80a, REQU82 one part is build of simple bodies , called primitives and using Boolean operators on them . As it is shown in Fig 2 elementary primitives are box , cylindar , cone and sphere . These are elementary primitives in Solid modeling technique . Using Boolean operators (union , difference and intersection) we can combined bodies on few ways , Fig 3.

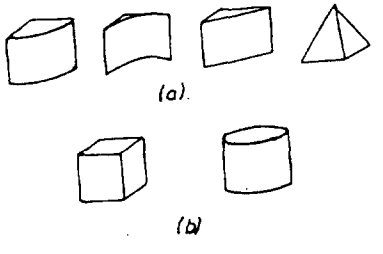


FIG 2 Elementary primitives

3.1 Representational schemes

Nowadays exist few representational schemes like Spatial enumeration , Cell decomposition , Constructive Solid Geometry (CSG) , Boundary representation (B-rep) , Sweep representations /REQU83 , REQU80a , REQU80b , MANT83a/ . We can introduce three most famous : CSG , B-rep and Sweep .

Constructive Solid Geometry (CSG) connotes

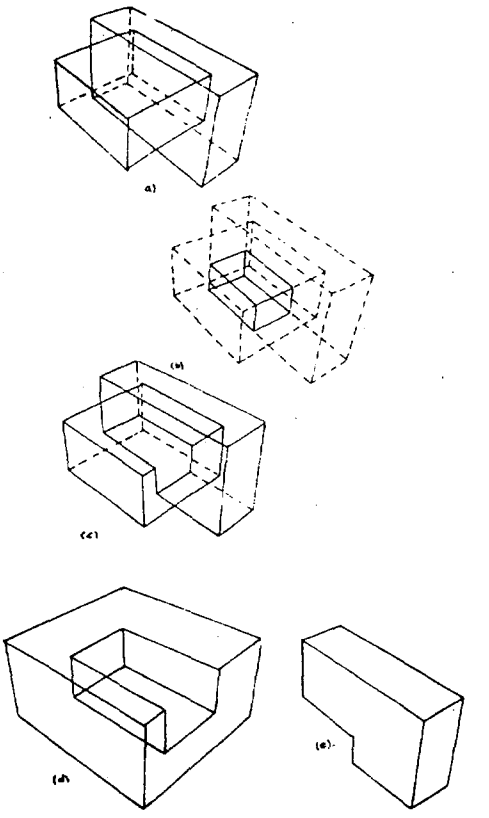


FIG 3 a) A non-physical "assembly" of two blocks , b)regularised intersections c)regularized union , d)regularized difference and e)other regularized difference

a family schemes for representing rigid solids as Boolean constructions or combinations of solid components via the regularised set of operators defined in the REQU80b and REQU80a . CSG and B-rep are the best understood and currently most important representation schemes for solids .

CSG representations are (ordered) binary tree . Nonterminal nodes represent operators , which may be either rigid motions or regularized union , intersection or difference ; terminal nodes represent primitive leaves which represent subsets of E^3 , or transformation leaves which contain the defining arguments or rigid motions . CSG trees may be defined by the following :

⟨CSG tree⟩ ::= ⟨Primitive leaf⟩/⟨CSG tree⟩ ⟨Set operator node⟩ ⟨CSG tree⟩ /⟨CSG tree⟩⟨Motion node⟩ ⟨Motion arguments⟩ (4)

The semantics of CSG-tree representations is clear (Fig 4).

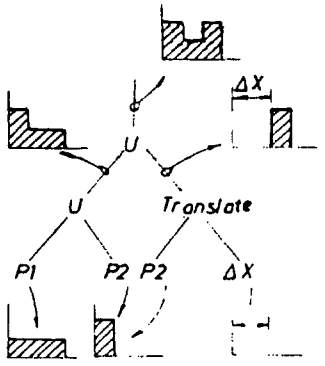


FIG 4 A CSG tree and the solids represented by its subtrees (solids are shown in ortographics projections)

Fig 4 and equation 4 shows that main component of each solid are : set of primitives , set of Boolean operators and set of transformation operators (Scaling , Translation and Rotation) , see NEWM81 and FOLE82 .

Most general way is halfspace model . Elementary halfspace is represented using real function

$$F(x,y,z) = ax + by + cz + d = 0 \qquad (5)$$

whuch has positive values outside halfspace , zero on the boundary and negative value in his inside . For example , solid cylindar has been represented using halfspaces which it's shown on Fig 5.

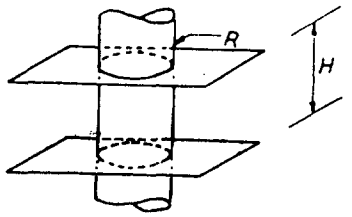


FIG 5 Representation cylindar using halfspaces. Object is constructed from one (unbounded) cylindar halfspace in form $x^2 + y^2 = R^2$ and two plane halfspaces form z=0 and z=H .

Second important schematic is B-rep . Boundary representation of solids are familiar to most computer scientists because of their use in computer graphics . A solid is represented by segmenting its boundary into a finite number of subsets usually called "faces" or "patches" , and representing each face by (for example) its bounding edges and vertices , plus data defining the surface in which the face lies , Fig 6 .

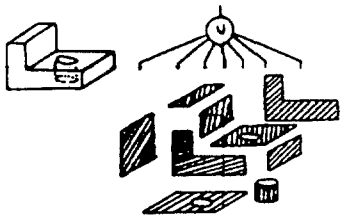


FIG 6 Boundary representation

B-rep is develope on i.e. Euler operators which are explained in next section .

The third unambiguous and popular scheme is called (simple) sweeping . The idea is that a "solid" set of points can be representad as the Cartesian product of an "area set" and "trajectory set" . (Intuitively , one percieves the area set as moving along the trajectory and "sweeping out" the represented solid). Fig 7a shows an "extruded" object represented via translatorial sweeping ; rotationally symetric objects can be represented by rotational sweeping . Fig 7b provided an example which combine sweeping and restricted form of set union called gluing .

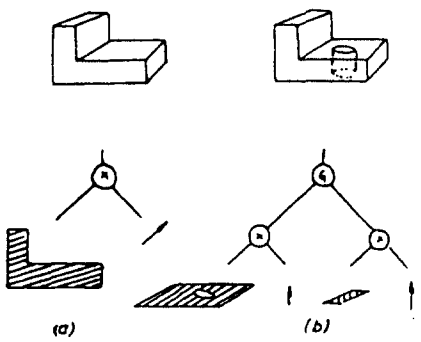


FIG 7 Pure (a) and hybrid (b) sweep representations

4 BASIC ELEMENTS OF TOPOLOGY AND EULER-POINCARE FORMULA

As we say , B-rep is build on topological concept . We can introduce basic topological definitions , shown in Fig 8 and in Appendix A.

Mathematically speaking models are described by Topological boundary . The boundary is represented by their bounding edges and vertices . It uses convention that faces need not simple connected but may include cavities ; the component of a boundary of a face are refered to as loops .

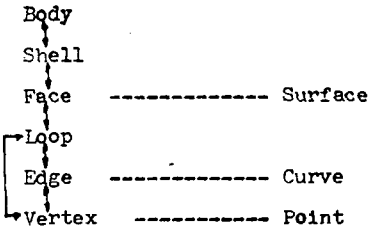


FIG 8 Hierarchy of topological entities and corresponding geometric equivalents

One of the loops represent the "outer" boundary of the face , while the others describe it's cavities . We shall use the term ring to denote an interior loop ; for instance , a face with one cavity is said to have one ring .

A boundary model might be represented simply as a set of multiconnected polygons . Each loop of each polygon would then be represented as a list of vertices . To speed up various operations on the model , boundary modelers frequently store other explicit data on connections between faces , loops , edges and vertices of solid . These data are commonly refered ta as the Topology of the model , while data such as face equotions or vertex coordinates are refered to as the geometry model /MILA82 , MANT82 , EAST79 , BRAI82 ,BAER79 , WEIL85/.

Euler operators derive their name from the well-known Euler's Law : In any simple polyhedron , the numbers of faces (F) , edges (E) and vertices (V) must satisfy the equation :

$$V - E + F = 2 \qquad (6)$$

The formula may be generalized to arbitrary solids by introducing three other parameters , namely (1) the total number of rings (cavities in faces , R) in the solid , (2) the total number of holes (H) through solids , and (3) the number of disconnected components (S) in a collection of solids . The general formula is

$$V - E + F = 2(S - H) + R \qquad (7)$$

As noted in BRAI82 just five operators (with their inverses) are sufficient for describing all objects satysfying the Euler formula . Let us consider the cube representad by six faces , twelve edges and eight vertices , as shown in Fig 9a.

Consider for a moment the effect of removing the edge marked by an arrow . In the resulting object , Fig 9(b) , the faces which meet at the edge are united and remaining collection of five faces , eleven edges and eight vertices, is generated a simpler model .

Performing the same operation on the edge marked in Fig 9(b) creates the object in Fig 9(c) . It has an edge belonging twice to the loop indicated by the dashes . The "upper" vertex of the edge is adjacent to no other edges . Another operation removes such a combination of an edge and a vertex and creates the model in Fig 9(d) . These two operators can remove all edges and all vertices except one . Athird operation can remove the remaining stripped-down , one vertex model .

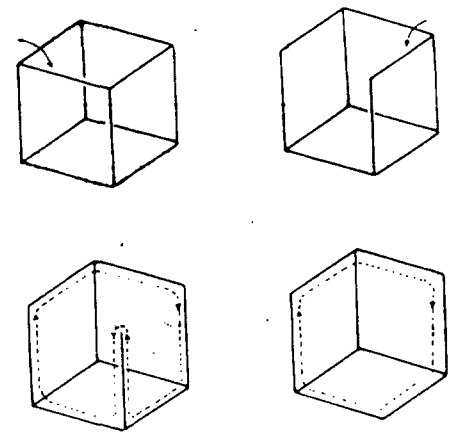


FIG 9 A cube (a) , after removal of one edge (b) , a second edge (c) , and vertex and third edge (d) . Note that each of the operators restores the validity of the generalized Euler form

The operations are examples of Euler operators. We call the three destructive operators "kef" , "kev" and "kvsf" for "kill edge , face" , "kill edge and vertex" , "kill vertex , solid and face" . Their inverses are "mef" , "mev" and "mvsf" where "m" stands for "make" .
Collection of operators which describes all models satisfy equation (7) is given in Table on Fig 10 .

TABLE I
Euler operators

Operator	Explanation
MVSF	Make vertex , solid , face
KVSF	Kill vertex , solid , face
MEV	Make edge , vertex
KEV	Kill edge , vertex
MEF	Make edge , face
KEF	Kill edge , face
KEMR	Kill edge , make ring
MEKR	Make edge , kill ring
KFMRH	Kill face , make ring , hole
MFKRH	Make face , kill ring , hole
SEMV	Split edge , make vertex
JEKV	Join edges , kill vertex

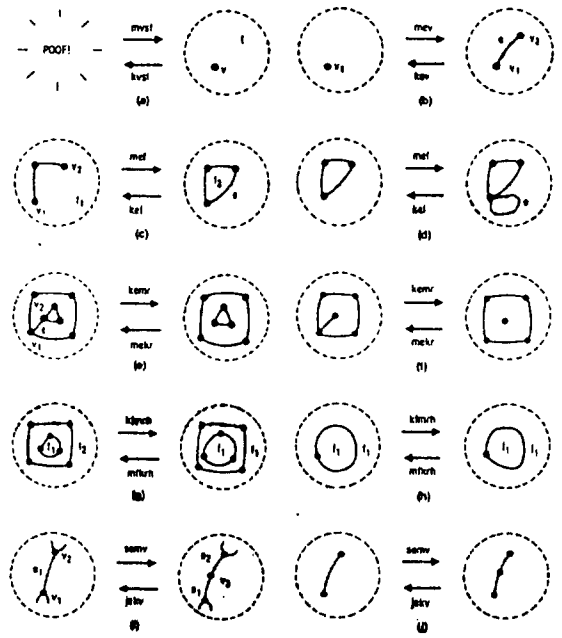


FIG 10 Euler operators :plane models

An example of body with hole , Fig 11 , shows manipulation with Euler operators .

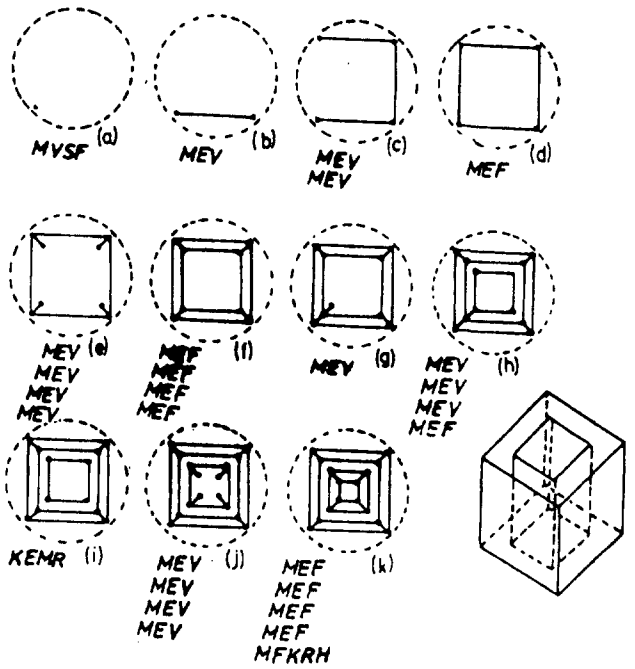


FIG 11 A sample definition (MANT82)

5 SURFACE MODELLING

Surface modeling emerged as a field of CAD in the early 1960s as result of work done by S.A. Coons at MIT , USA in representing mathematically the shape of sculptured objects such as aircraft exterior skins , ship hulls and car bodies .

The mathematical methods are polynomial functions , which allow through differentiation easy determination of geometric properties of the surface tangents , normals and curvature . Polinomial of high degrees require a large number of coefficients . Bernstein - Bezier or B-spline methods requires only 16 control points . The mapping from control points to actual surface is done through a 4x4 matrix which for Bezier representation has following form :

$$P(u,v) = UBCB^{T}V^{T} \qquad (8)$$

where P(u,v) is the point in Cartesian space which corresponds to the parametric surface point (u,v) , and where :

$$B = \begin{vmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{vmatrix} \qquad (9)$$

is the Bezier basis matrix for a bicubic patch , where

$$C = \begin{vmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ c_{21} & c_{22} & c_{23} & c_{24} \\ c_{31} & c_{32} & c_{33} & c_{34} \\ c_{41} & c_{42} & c_{43} & c_{44} \end{vmatrix} \qquad (10)$$

is the set of control points , which are arrnnged on topologically rectlinear grid according th their subscripts

$$U = \begin{bmatrix} 1 & u & u^2 & u^3 \end{bmatrix} \qquad (11)$$

$$V = \begin{bmatrix} 1 & v & v^2 & v^3 \end{bmatrix} \qquad (12)$$

are the parametric basis vectors .

The modelling of a patch amounts , therefore , to the manipulation of its 16 control points in thr 3D Cartesian space .

6 THE REPRESENTATION OF POLYHEDRAL SOLIDS

Most of models developed so far mathematically representing a solid object are based on the general formal definition of its shape as a set of contigous points in 3-space . The idea that relations between elements of list is topological is shown on Fig 8 and 12 , /KALA83b , WIED85/. Body is build of faces . The two-axis (or more) curvature discontinuity points are known as vertices . The volume enclosed by this set of elements is known as polyhedral shape .

It is possible to implement a polyhedral shape representation by means of a hierarchical data structure , such as the one shown in Fig 12 . In this scheme the nodes represent one shape element each , and the arcs represent the relations between them . As it shown in BAER79 and WEIL85 it is possible to get one node for base . For example , a polyhedral shape bounded

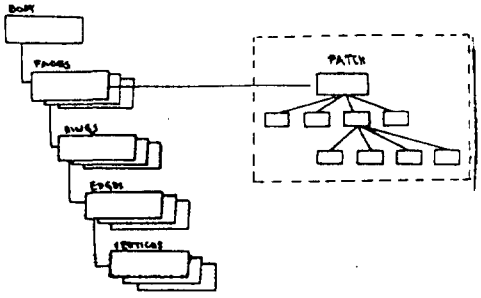


FIG 12 A hierarchical representation of the shape model (polyhedral representation without sculptured surface including is on the left side)

by planar surfaces can be represented by means of two types of nodes - faces and vertices , when the arcs between them are constructed such that each string of successive vertices represents a ring of edges bounding face . Another representation has been deviced by Baumgart /BAUM 75/ . Other types of structures are described in BRAI82 and WEIL85 . This example is get from KALA83b and NEWM81 and demonstrate representation in "winged-edge" structure . This scheme is derived from a proof to Euler Law relating the number of various elements in a polyhedral shape model (Eq 6 and 7).

Since an edge is adjacent to exactly two faces , it is component in two rings which bound one face each . Assuming these rings are oriented , the edge has a predecessor and a successor each , as well as two bounding vertices (Fig 13) .

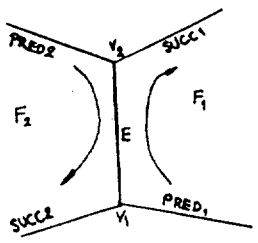
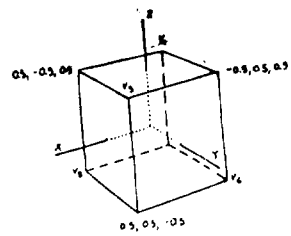


FIG 13 The "winged-edge" data structure

Using the "winged-edge" structure together with its implications on the ring and face elements as the key for representing the shape , Baumgart obtained a structure which is extremly efficient for manipulation purposes a set of operators , for manipulating a structure . (For further informations in data base for boundary representation see BRAI82).

7 A RELATIONAL MODEL FOR REPRESENTING POLYHEDRAL SHAPES

The relational database model is one of the three most common data definition languages , the other two being the network and the hierarchical models . Its underlaying mathematical foundation derives from the set-theoretic concept of relation . The object called relations , are commonly represented in a 2D tabular form . Each relation corresponds to a single class of data , which in the case of the polyhedral shape can be faces , rings , edges and vertices . The rows of relation are called tuples . Each tuple identifiable through a unique key , and its columns are called domains ; each domain identifiable through a unique name . The number of domains in a relation determines its degree . For example , in the edge relation (Fig 14) , each tuple correspond to a single edge , and its domains correspond to one of the edge adjacent items , such as its predecessor edge on each of the two rings in which it is a member .



Face list

Face no	Ring on F
1	1
2	2
3	3
4	4
5	5
6	6

Ring list

Ring No	Next R/F	Face
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6

Vertex list

Vertex No	Xcoord	Ycoord	Zcoord
1	-0.5	-0.5	0.5
2	-0.5	-0.5	0.5
3	0.5	0.5	0.5
4	0.5	-0.5	0.5
5	-0.5	-0.5	-0.5

6	0.5	0.5	-0.5
7	0.5	0.5	-0.5
8	0.5	-0.5	-0.5

Edge list

EdgNo	Rngl	Esucl	Eprdl	Vertl	Rng2	Esuc2	Eprd2	Vrt2
1	1	2	4	1	2	9	10	2
2	1	3	1	2	3	10	11	3
3	1	4	2	3	4	11	12	4
4	1	1	3	4	5	12	9	1
5	6	8	6	6	2	10	9	5
6	6	5	7	7	3	11	10	6
7	6	6	8	8	4	12	11	7
8	6	7	5	5	5	9	12	8
9	5	4	8	5	2	5	1	1
10	2	1	5	6	3	6	2	2
11	3	2	6	7	4	7	3	3
12	4	3	7	8	5	8	4	4

FIG 14 A relational representation of the polyhedral shape model

8 EXTENSION OF SOLID MODEL USING SOLID OFFSETING

Offset solids are expanded or contracted versiona of an original objects . To solid offset (abbreviated s-offset) a solid S by a positive distance r one adds to the solid all points exterior to S that lie within a distance r of the boundary of S . For a negative s-offset one subtracts from the solid all the points of S within a distance r from its boundary . Positive and negative s-offsets for a simple L-shaped "2D solid" are shown in Fig 15 /ROSS86/.

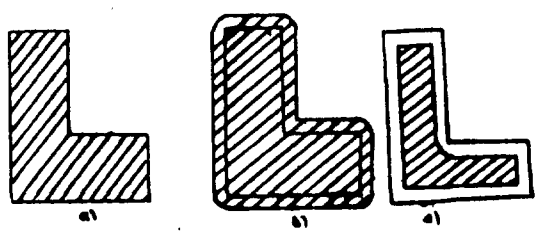


FIG 15 A simple L-shapped object (a) a positive offset (b) , and a negative offset (c)

In NC and related applications one offsets smooth surfaces by "moving along the normal" by a distance r to generate other surfaces and offsets planar curves similarly to generate other curves . (This operation is refered as normal offseting or simply n-ofseting to distinguish it from solid offseting) . The current understanding of normal offseting is sumarized in TILL84 and FARO85 . Briefly , there is no accepted mathematical definition of n-offseting may led to cusps and self-intersections . Heuristic approaches proposed in TILL84 provide reasonable results in many but not all of the possible cases .

It is interesting that offseting has many potential applications in :

- Tolerance analysis . Three-dimensional tolerance zones may be defined by offsets , and used to determine whether mechanical parts satisfy specified tolerances .
- Approximate object equality . Two solids X and Y are equal within some specified resolution ε. The solids X and Y may be for example , a desired part and the part generated by a sequence of machining operations .
- Physical process modelling . Processes such as coating and etching may be modelled , at least to a first approximation , by expanding and shrinking operations .
- Cutter path generation . Computing cutter paths for NC machining by s-offseting solids seems much more usefull than current n-offseting techniques , because it does not produce self-intersecting paths .

9 EXTENSIBILITY OF THE RELATIONAL DATABASE MODEL

Aspects shown in section 8 and posibility to have an information abuout surface finish , heat treatment and material type of each face of model give us an idea how to extend this relational model . Looking from technological aspects it is possible to make a link between a faces in hierarchical scheme and a state of machininig(surface finish) of each face . This link is shown on Fig 16 .

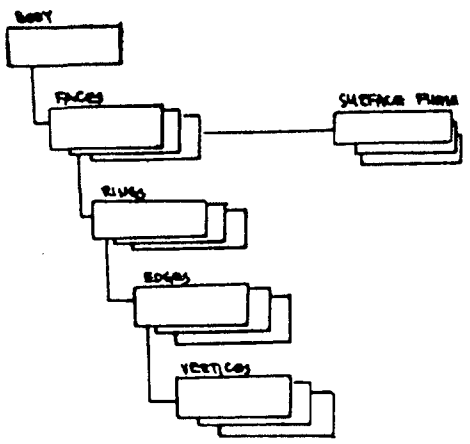


FIG 16 Extension of representation of polyhedral shape model

Correspondence between face list of surface finish is one to one . This introduce , very easy in Computer Intelligent Process Planning becau se we have informations about state of part sur-

face on one hand and on second other perimeters like a necessary tools , machine tools and so on /PHILL85/.

10 CONCLUSION

The presentation of the characteristics of an integrated modelling system in this paper is the first step in attempting to bridge the gap between some of the desired levels of design capabilities in a comprehensive manner rather than local extensions , that will allow the representation and manipulation of solid objects . As it is shown basic part of modul is database concepted on relational schemes . This database is very flexible and has application in other areas . (Here is not shown but solid model play a big role in immage processing in robotics) . One of the interesting fields is CAPP systems . Solid database on relational principles is main information about body "immage" . This data in such a relation model of the shape can be structured in many ways and expanded into a suitable manipulative structures when necessary .

APENDIX A

Vertex	Aunique point in a space . A vertex ' lies in one or more faces .
Edge	A finite nonself-intersecting , directed space curve bounded by two not necessarily distinctc vertices . An edge lies in at least one and not more than two faces .
Face	A finite connected , nonself-intersecting , oriented piece of surface bounded by one and more loops . A face lies in a single shell .
Loop	An oriented ordered alternating seqence of vertices and edges . A loops are nonself-intersected , and lies in single face , and form a bound of face
Shell	The collection of consistently oriented faces forming the bound of a single connected , closed volume .

REFERENCES

ALAG84 Alagic Suad :"Relational Database" , Svjetlost , Sarajevo , 1984 (on Serbocroat)

BAER79 Baer A , Eastman C , Henrion M : "Geometric modelling : A survey" , CAD , Vol 11 , No 5 , September 1979 , pp 253 - 272

BAUM75 Baumgart Bruce : " A polyhedron representation for computer vision " , AFIPS Proc , Vol 44 , 1975 , pp 589-596

BRAI82 Braid I. C, Hillyard R. C , Stroud I. A : "Stepwise construction of polyhedra in geometric modelling" , Ed. K. W. Brodlie : "Mathematical methods in computer graphics and design" , Academic press , London

EAST79 Eastman Charles , Weiler Kevin :"Geometric Modeling using the Euler operators " , Proceeding First Annual Conf. on Computer Graphics in CAD/CAM systems , MIT , April 1979 , pp 248-259

FARO85 Farouki R. T.:"Exact offsets procedures for simple solids" , Computer Aided Geometric Design , Vol 2 , No 4 , December 1985 , pp 248-259

FOLE82 Foley D.James , Van Dam Andreas :"fundamentals of Interactive Computer Graphics" , Addison Wesley , 1982 ,

KALA83a Kalay Yehuda :" Modelling polyhedral solids bounded by multi-curved parametric surfaces" , CAD , Vol 15 , No 3 , May 1983 , pp 141+146

KALA83b Kalay Yehuda : " A relational database for nonmanipulative representation of solid objects " , CAD , Vol 15 , No 5 , September 1983 , pp 271 +276

KOCO( ) Kočović Petar : "Geometric Modelling using Euler Formulas" , (to be appear in Informatica , serbocroat version)

MANT82 Mäntylä Märtti, Sulonen Reijo :"GWB :A Solid Modeler with Euler operators", IEEE Computer Graphics and Applications september 1982 , pp 17+31

MANT83a Mäntylä Märtti :"Solid Modelling : Theory and Applications", Pre-Conferen ce Tutorial , Geometric Modelling ,EUROGRAPHICS'83 , Zagreb , August 1983

MANT83b Mäntylä Martti : "Computational Topology : A Study of Topoloogical manipulations in Computer Graphics and Geometric Modelling" , ACTA Polytechnica Scandinavica , No 37 , 1983

MILA82a Milačić Mariana : " Development and application of a mathematical method in the decomposition of multilevel hierarchical systems with an information base " , Ph.Doc.Disertation , University of Belgrade , 1982 (on Serbocroat)

MILA82b Milačić Vladimir :"Computer-Based Informatization on Manufacturing Engineering Activities" , Int . J. Prod . "es , Vol 20 , No 3 , pp 362-403

MILA84 Milačić Vladimir , Kalajdžić Milisav: "Logical structure of manufacturing process design - fundamentals of an expert system for manufacturing process planning" , 16 CIRP International Seminar on Manufacturing Systems , To-

kyo , 1984 , pp 93+101

MILA85 Milačić Vladimir :"SAPT-Expert System of Manufacturing Process Planning" , Winter Annual Meeting of ASME , Miami Beach , November 1985 , pp 43+54

NEWM81 Newmann William , Sproul Robert : "Principles of Interactive Computer Graphics" , McGraw-Hill International 1981

PHIL85 Phillips R.H , Arunthavathan V , Zhou D.X :"Symbolic representation of CAD data for Artificila Intelligence based Process Planning " , Winter Annual Meeting of ASME , MIami Beach , November 1985 , pp 31+42

REQU80a Requicha A.G. Aristides : " Representation of Rigid Solid Objects" , Lecture Notes in Computer Science (Edition J. Encarnacao) , Springer Verlag , pp 2+78

REQU80b Requicha A.G. Aristides : "Representations for Rigid Solids Theory , Methods and Systems" , ASM Computing Survays, Vol 12 , No 4 , December 1980 , pp 437+464

REQU82 Requicha A.G. Aristides , Voelcher B. Herbert : "Solid Modeling : A hisTorical summary and contemporary asse ssment" , IEEE Computer Graphic and Applications , March 1982 , pp 9+24

REQU83 Requicha A.G. Aristides , Voelcher B. Herbert :"Solid Modelling : Current Status and Research directions" , IEEE Computer Graphics and Applications , October 1983 , pp 25+37

ROSS86 Rossignac R. Jaroslaw , Requicha A.G. Aristides : "Offseting operations in solid modelling" , Computer Aided Geometric Design , Vol 3 , No 2 , August 1986 , pp 129+148

TILL84 Tiller Wayne , Hanson Eric : " Offsets of two-dimensional Profiles" , IEEE Computer Graphic and Applications , Septembar 1984 , pp 36+46

VOEL77 Voelcker B. Herbert , Requicha A.G. Aristides : "Geometric Modelling of Mechanical Parts and Processes" , Computer , December 1977 , pp 48+57

WEIL85 Weiler Kevin : "Edge-Based Data Structures for Solid Modeling in Curved Surface Environments", IEEE Computer Graphics and Applications , January 1985 , pp 21+40

WIED85 Wiederhold Gio : "Database Design", McGraw-Hill International , 1985

IMPLEMENTING PASCAL-LIKE CONSTRUCTS: AN EXERCISE IN PROLONG PROGRAMMING

INFORMATICA 2/87

Bogdan Filipič
Department of Computer Science and Informatics
»Jožef Stefan« Institute, Ljubljana

UDK 681.3.06:519.682 PROLOG

ABSTRACT. Due to the declarative meaning of programs, Prolog is a powerful programming language. However, in practice it turns out that numerous tasks within a certain kind of programs are quite procedural. The paper describes a simple implementation of some Pascal-like construcs in Prolog as a practical solution to this problem.

IMPLEMENTACIJA PASCALSKIH KONSTRUKTOV KOT VAJA IZ PROGRAMIRANJA V PROLOGU. Prolog je zaradi deklarativnega pomena programov močan programski jezik, vendar se v praksi izkaže, da so mnoga opravila znotraj določenih programov povsem proceduralnega značaja. V članku je kot praktična rešitev tega problema prikazana implementacija nekaterih pascalskih konstruktov v Prologu.

INTRODUCTION

Due to the declarative meaning of programs, Prolog is a powerful programming language [1,3]. It is especially well suited for solving non-numerical problems. From the programmer's point of view, programming in Prolog is very efficient. On the other hand, Prolog implementations suffer from the lack of supporting the procedural programming approach. In practice it namely turns out that numerous tasks within a certain kind of programs are quite procedural. One may, for example, wish to perform an action conditionally, repeat a procedure until a certain condition is satisfied or execute a sequence of actions a given number of times. To do this in Prolog, we usually use cuts and repeat-fail loops. Unfortunatelly, improving efficiency through these mechanisms often weakens the clarity and the conciseness of a Prolog program.

The problem can be solved neater by using explicit definitions for the control of execution. Including Pascal-like constructs into Prolog may be seen as a practical solution to this problem.

IF-THEN AND *IF-THEN-ELSE* PROCEDURES

Some Prolog implementations support conditional by default. Remember, for example, *P -> Q ; R* from DECsystem-10 Prolog [2], C-Prolog [5] or Quintus Prolog [6]. Arity Prolog [7] even supports *ifthen/2* and *ifthenelse/3* predicates. Here are common definitions of these procedures:

```
if_then(P,Q) :- call(P), !, call(Q).
if_then(P,Q).

if_then_else(P,Q,R) :- call(P), !, call(Q).
if_then_else(P,Q,R) :- call(R).
```

Using operators, we may simply define an appropriate predicate *if/1* that is determined as a principal functor. Its definition includes both above alternatives (see implementation in the appendix). We may now code conditionals of both forms:

```
if P then Q.
if P then Q else R.
```

where *P*, *Q* and *R* denote single Prolog goals or sequences (i.e. conjunctions or disjunctions) of goals.

CASE PROCEDURE

Once the *if* predicate has been implemented, we may define *case* as a sequence of *if* procedures trying to match a certain condition value and to satisfy related goal(s). We introduce the Prolog *case* construct

```
case X of [ X1:Q1, X2:Q2, ..., Xn:Qn ].
```

that is interpreted as

```
if X = X1 then Q1
else if X = X2 then Q2
else if   .
          .
          .
else if X = Xn then Qn.
```

and

```
case X of
   [ X1:Q1, X2:Q2, ..., Xn:Qn otherwise R ].
```

that stands for

```
if X = X1 then Q1
else if X = X2 then Q2
else if   .
          .
          .
else if X = Xn then Qn
else R.
```

Note that our implementation does not support more than one value being assigned with each alternative. This can be done by joining possible values into a list and substituting condition `X = Xi` with testing the list membership. The following is an example of using the *case* construct:

```
menu_selection :-
   write( 'Selection? ' ),
   read_line( Answer ),
   case Answer of
      [ a : option1( ... ),
        b : option2( ... ),
        c : option3( ... ),
        h : ( help, menu_selection),
        x : exit_menu
        otherwise
           ( beep,
             write( 'Illegal answer' ), nl,
             menu_selection
           )
      ].

   read_line( Answer ) :-
      get( ASCII ),
      name( Answer, [ASCII] ).

   beep :- put(7).
```

REPEAT-UNTIL PROCEDURE

Here we introduce the *repeat/1* predicate that will have similar effect as the built-in *repeat/0*. You must keep in mind that proposed

```
repeat Q until P.
```

is just a Pascal-like notation for the procedure that will be executed in the Prolog sense, i.e. failure of *Q* will result in backtracking. The execution may be viewed as Pascal-like when backtracking is not possible, for example:

```
repeat ( write('Filename? '), read(F) )
until exists(F).
```

Note that, when *Q* stands for a sequence of goals, they must be put into brackets and the left bracket must be separated from the principal functor repeat by blank. Otherwise the entire expression denotes a predicate of an arity greater than 1 which will fail, of course. Similarly, this syntax restriction must be considered when using *if*.

FOR PROCEDURE

In order to implement the *for* loop, let us first introduce a utility for managing global counters [4]. Suppose we have certain values in our program that can be passed to or modified by any part of the program. Modifying these values may be understood as assigning values to global variables in procedural languages. Such variables are particularly suitable as counters. Each counter is specified by its name, a key, and the related integer value. The counter managing predicates below simply use *retract* and *assert* to assure the current value of a counter to be recorded in the database.

Based on previously defined *repeat* procedure and the global counters, the *for* loop has the following two forms:

```
for [Count,I] := I1 to I2 do Q.
for [Count,I] := I1 downto I2 do Q.
```

Count identifies the procedure and should be instantiated to a Prolog constant. Furthermore, the global counter with the key *Count* is activated when executing the procedure. Its current value is instantiated to *I*. The following example illustrates how to use the loop:

```
for [i,I] := 1 to 10 do
   ( write(I), nl ).
```

Again, the execution is Pascal-like only when the goal(s) appearing in the loop can be satisfied in no more than one way.

CONCLUSION

The paper presents the implementation of some Pascal-like constructs in Prolog. We found them useful for writing procedural segments of programs, such as input and output procedures. As already stated, our purpose is not to reduce the importance of standard Prolog concepts, but just to add some convenient procedural features. In our opinion, combination of both declarative and procedural approaches is the right solution when wondering about how to code a complex task effectively.

And finally, the reader might have noticed that we said nothing about the *repeat* procedure when discussing Pascal-like features. The reader is invited to implement it himself as an exercise.

REFERENCES

[1] Bratko I.: Prolog Programming for Artificial Intelligence, Addison-Wesley, 1986

[2] Byrd L., Pereira F., Warren D.: A Guide to Version 3 of DEC-10 Prolog, Department of Artificial Intelligence, University of Edinburgh, 1983

[3] Clocksin W.F., Mellish C.S.: Programming in Prolog, Springer-Verlag, 1984

[4] Filipič B., Mozetič I.: A Library of Prolog Utilities, Report IJS DP-4466, Jožef Stefan Institute, Ljubljana, 1986

[5] Pereira F.: C-Prolog User's Manual, University of Edinburgh, Department of Computer Aided Architectural Design, 1984

[6] Quintus Prolog User's Guide and Reference Manual, Quintus Computer Systems Inc., Palo Alto, 1985

[7] The Arity/Prolog Programming Language, Arity Corporation, Concord, 1986

APPENDIX: THE PROGRAM

```
%-----------------------------------
%  File PROCED :
%  Implementing Pascal-like constructs
%-----------------------------------

:- op( 900, fx,  if   ).
:- op( 850, xfx, then ).
:- op( 800, xfx, else ).

:- op( 900, fx,  case      ).
:- op( 850, xfx, of        ).
:- op( 800, xfx, otherwise ).
:- op( 750, xfx, ':'       ).

:- op( 900, fx,  repeat ).
:- op( 850, xfx, until  ).

:- op( 900, fx,  for    ).
:- op( 850, xfx, to     ).
:- op( 850, xfx, downto ).
:- op( 800, xfx, do     ).
:- op( 750, xfx, ':='   ).

:- [counts].   % Counter management.

if P then Q :-
   if_then_else(
      Q = (R else S),      % If 'else' found
      if_then_else(P,R,S), % then if_then_else
      if_then(P,Q)         % else if_then.
   ).

   if_then(P,Q) :- call(P), !, call(Q).
   if_then(_,_).

   if_then_else(P,Q,_) :- call(P), !, call(Q).
   if_then_else(_,_,R) :- call(R).

case X of [ Xn:Q otherwise R ] :-
   if X=Xn then Q else R, !.
case X of [ Xn:Q ] :-
   if X=Xn then Q, !.
case X of [ Xi:Q ; Others ] :-
   if X=Xi then Q else case X of Others.


repeat Q until P :-
   repeat, call( (Q, !) ), call(P).

for [_,_]:=I1 to I2 do _ :-
   I1>I2, !.
for [Count,I]:=I1 to I2 do Q :-
   ctr_set(Count,I1),
   repeat ( ctr_inc(Count,I), Q )
   until I=I2,
   ctr_remove(Count), !.
for [_,_]:=I1 downto I2 do _ :-
   I1<I2, !.
for [Count,I]:=I1 downto I2 do Q :-
   ctr_set(Count,I1),
   repeat ( ctr_dec(Count,I), Q )
   until I=I2,
   ctr_remove(Count), !.
```

```
%---------------------------
%  File COUNTS :
%  Managing global counters
%---------------------------

% ctr_set/2 sets a counter to the
% desired number.

ctr_set(Key,N) :-
   integer(N),
   ctr_remove(Key),
   assert( counter(Key,N) ), !.

% ctr_inc/2 increments a counter and
% returns its previous value.

ctr_inc(Key,N) :-
   retract( counter(Key,N) ),
   N1 is N + 1,
   assert( counter(Key,N1) ), !.

% ctr_dec/2 decrements a counter and
% returns its previous value.

ctr_dec(Key,N) :-
   retract( counter(Key,N) ),
   N1 is N - 1,
   assert( counter(Key,N1) ), !.

% ctr_is/2 returns the current value
% of a counter.

ctr_is(Key,N) :-
   counter(Key,N), !.

% ctr_remove/1 removes a counter
% from the database.

ctr_remove(Key) :-
   retract( counter(Key,_) ), fail.
ctr_remove(_).
```

LANGUAGE CONSIDERATIONS OF PARALLEL PROCESSING SYSTEMS
PART ONE: Concurrent microprocessing systems

INFORMATICA 2/87

UDK 681.3.06:519.682/.683

Peter Kolbezen
Institut »Jožef Stefan«, Ljubljana

ASTRACT - Full parallelism offered by the multi-processor is not still fully exploited. Much work that has been done in structured programming to separate a mono-processor program into well-defined modules, and attempts to systematize the interactions between modules, have helped to achieve a more disciplined approach to software development with much benefit to multi-mikroprocessor software.

This paper presents various issues relevant to language aspects of parallel processing systems. The objective is to present a discussion of issues and some of the current approaches rather than a well-developed metodology of software, which has yet to be developed. New approaches to parallel processing architecture are briefly outlined too.

O JEZIKIH SISTEMOV PARALELNEGA PROCESIRANJA. PRVI DEL: Konkurenčni mikroprocesorski sistemi. Popolna sočasnost, ki jo omogoča materialna oprema večprocesorskih sistemov, še ni dovolj izkoriščena. Da bi se ta cilj dosegel, je bilo med drugim vloženo že veliko napora tudi v strukturirano programiranje, ki deli enoprocesorski program v dobro definirane module, poskuša sistemizirati akcije med moduli, pomaga doseči bolj urejen pristop k razvoju programske opreme in ji daje številne prednosti.

Članek podaja zaključke, ki izhajajo iz jezikovnega vidika na sisteme paralelnega procesiranja. Obravnavani so zgolj rezultati in novejši poskusi reševanja problemov programske opreme. O kaki bolj dovršeni metodologiji programske opreme pa ni mod govoriti, saj je le-ta še vedno v razvojnih fazah.

Na kratko so opisane tudi nekatere najvidnejše računalniške arhitekture, ki še posebej učinkovito podpirajo paralelno procesiranje.

1. INTRODUCTION

High level languages and their translators have become essential for writing application programs for mono-processor systems. The same, however, cannot be said for multi-microprocessor systems. The immense variety of applications and hardware architectures, and the diversity of philosophies about how systems shoud be structured, makes it extremely difficult to design languages that are likely to be widely accepted. It still remains a difficult challenge to design a high level language which is sufficiently general and modular to accommodate a large number of architectural types of machines /1/. In the absence of bold and fresh ideas to express concurrency, it is then natural that current thinking is along the lines for extending or generalizing the sequential programming languages /2/. At least it is known that using this approach one has something that works for an isolated microprocessor which forms a constitutent part of the whole system. Thus a sequential language enables individual software modules to be written. This is a rather primitive approach, however, where concurrency (which requires a control and communication structure), synchronization for resource sharing, efficiecy and robustness aspects are outside the language consideration.

A further difficulty stems from the fact that the language issues and runtime support aspects cannot be isolated totally. The attributes of the kernel are important in deciding whether or not certain issues need to be dealt with at the language level.

Most of the language proposals in the concurrent programming area also have an underlying model of distributed computing. The many of these languages are in the research phase and any have not been implemented, also there is little hard practical experience. Most of the time the underlying model is not explicitly stated.

Event if one attempts to extract the underlying model from a proposal, it is not always an easy task. Sometimes the model and languages issues become inseparable. The choice of the model would affect the programming methodology and the proof techniques for a language based on that model. A model provides a conceptual framework in which to discuss and understand the behaviour of concurrent computations, and is intended to capture the underlying philosophy of a programming language.

A high level language is a medium which not only enables us to obtain a machine executable code but, perhaps more importantly allows us to formulate an application precisely. In this sense, there is a greate vacuum for a vehicle to describe concurrent applicatios formally.

Another difficulty in using languages applicable to multi-microprocessor systems is the necessity for a translator. Translator writing immediately requires the specification of the target machine. It is desirable that the translator also runs on the target machines. Since there is no architectural uniformity, this requires a translator design which is capable of running on widely varying configurations. Ideally, a translator also should take advantage of the structure and hence be modular. This requires significant departure from compiler writing for mono-processor systems.

2. FEATURES OF CONCURRENT LANGUAGE

Some of the desired features of a concurrent language can be listed as follows /3/:

- expressive power or richness - provability - ease and efficiency of implementation - easy of use - readability of resulting programs - impact of changes - extent of concurrency possible

Expressive power or richness This refers to the ability of the model/language in being able to express certain behaviours, i.e. the richness to be able to model certain computations like recursion, non-determinism, and so on. This property is also referred to as completeness or adequancy. An increase in expressive power is likely to be accompanied by an increase in the difficulty of proving programs. While it is desirable to have simplicity as one of the goals, it is not advisable to have that as the overriding criterion.

Provability One may be interested in proving many properties, like partial correctness, freedom from deadlocks, termination, fairness, etc. The presense of some constructs would make it extremely difficult, if not impossible, to prove certain properties. For example, at the current state of the art of program proving, the presence of time-outs could make the achievement of the tractability of proofs almost impossible. Of course, an important consideration is the power of the language used for specifyng assertions about program properties. The assertion language or the logic used should be rich enough to be able to specify formally various desired properties /4/.

Formalization of the semantics of constructs is an important prerequisite for program proving. While researches have been discussing all of the above properties for a long time, there are very few well defined techniques or formal methods to illustrate the existence of the necessary properties.

Ease and efficiency of implementation The implementation of certain features may be quite difficult to achieve. It is not sufficient merely to define primitives whose functionality makes them worth implementing. It must also be possible to deliver that functinality with reasonable efficiency. In most aplications the efficiency, or costliness, is likely to be an important consideration. While some constructs might be implemented easily, the efficiency of such implementations may not necessarily be good. The practicality of mechanisms would be measured by the efficiency of their implementations.

Ease of use The presence of powerful features does not mean that they would be easy to use. Normally high level constructs and good abstractions capabilities make thing easier. Ease of use and expressive power are complementary criteria. A model/language being rich enough to express a certain type of computation does not automatically mean that it could be done in an easy way-certain ingenious, awkward and obscure ways have to be resorted to. Constructs which reflect intuitive ways of abstractions would be appealing to the user.

While writing programs, language primitives should allow coherent combinations. Avoiding subtle interactions among primitives would make them easier to use and help reduce errors. The flexibility of the constructs is also an important factor in the ease of their use.

Readability of resulting programs Any proposal for new language features should be scrutinized closely to determine the effect of the proposed facility on program structure. The mechanisms should be such that they discourage complex and confusing structures. The presense of high level and very powerful constructs could lead to easily comprehensible programs. Of course, this may not always be the case. The ability to compose the process structure hierarchically should be of great benefit. In general, constructs that are easily verified are likely to be easily understood.

Impact of changes If the constructs do not include or force a high degree of modularity, a change in the definition of one process may necessitate many changes throughout the rest of the system. This would be highly undesirable, particularly if the number of the processes involved is quite large. Permitting a great degree of autonomy in the definition of processes would help a good deal in reducing and localizing the impact of changes.

Extent of concurrency possible The greater the degree of concurrency the constructs permit to be expressed, the better. But the overheads involved in supporting such concurrency should not be such as to offset the advantages gained through the increase in parallelism.

3. HIGH PARALLEL PROCESSING ARCHITECTURES

It is agreed by all concerned that the key to fifth generation computer architectures is a much higher degree of parallelism than is incorporated into computers at present. It is likely that there will be a number of layers of parallelism: closely coupled processing elements reflecting the parallelism inherent in inference or knowledge base processing operations, looser coupling between the various subsystems in a fifth generaton computer, and distributed processing across local and wide area networks of computers /3/.

At present there are two types of close-coupled parallelism implemented in computers: processing arrays and pipelines. Processing arrays are vectors of identical processing elesents which act synchronously to perform identical opreations on arrays of data. Pipelines are used for multi-stage operations /7/ such as floating-point multiplications, where each element of the pipeline carries out one step of operation, and passes its intermediate result

to the next element. Operations on successive sets of data can take place at intervals of one step. Parallel processing of this type, known as "regular" parallelism, will undoubtedly find a place in fifth generattion computers, but mechanisms to deal with irregular parallelism are the main topic for research. Three approaches are present today: parallel control flow, dataflow and graph reduction /9,10/.

Traditionally, by parallel control flow each step of a program is executed in sequence, under the control of a single program counter which determines the lowlevel operation to be carried out next. The flow of control is implicit in the structure of the program. Each statement in the module is a call to a more detailed processing procedure. Therefore, if a parallel computer system and programming language were available, the processing procedure are called at the same time. They executed in parallel, and the control module waits until all processing procedure are complete before continuing. Programming languages such as concurrent Pascal and Ada have facilities for operations of this sort computers, but it remains to be seen whether this approach, which is only a slight variation on conventional sequential processing, will be adequate for the radical demands of fifth generation architectures.

For a number of reasons, one of the most promising architectural models, certainly for the inference processing subsystems of a fifth generation computer, is dataflow architecture. It can cope with irregular as well as regular close-coupled parallelism, it is flexible and extensible, it has the potencial for very high data throughputs, and it reflects, at hardware level, the type of parallelism inherent in inference processing. The central idea of dataflow architecture is: a network of processing elements is set up, which reflects the logical structure of the task to be carried out, and items of data flow between the elements. Each elements operates at its own pace, and waits until it has a complete set of intermediate inputs before it "fires". There are two techniques for the control of such a network. In the totally data-driven approach, each element waits passively for data to arrive, whereas in the demand-driven regime each element issues requests "upstream" for data when it is ready for it. In general a dataflow computer or computer subsystems has three requirements:

- to store representations of program graphs,
- to implement some form of data tokens to flow through the graphs, and
- to provide suitable instruction processing facilities.

Each requirement poses certain problems, some of which are quite severe. Program graphs in practice will contain hundreds of thousands, if not millions, of arcs and nodes, and may not always reduce to the neat tree structure. Furthemore, if, as is almost certain, program contains recursive definitions, portions of the structures will be re-entrant. In this example the graph (recursive program graph) of this inference needs to replicate itself repeatedly during processing. Further, most of the data processed in knowledge-based systems does not consist of single items, but of large structures which would cause unacceptable overheads if they were passed through a dataflow network in their entirety. This problem is being approached by using pointers to the data structures in the dataflow networks, and accessing the structures from fixed memory only when they are required.

Three lines of research are being followed in response to these difficulties. The first is to regard a dataflow task as fixed at compile time, and to prohibit re-entrat code /12,13/. This static approach is ilustrated in Fig.1, which uses a network of binary processors each with two alternative output channels.

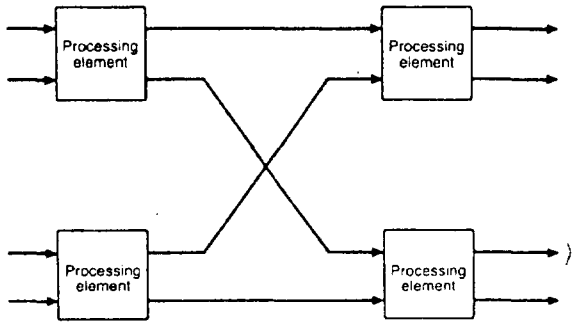
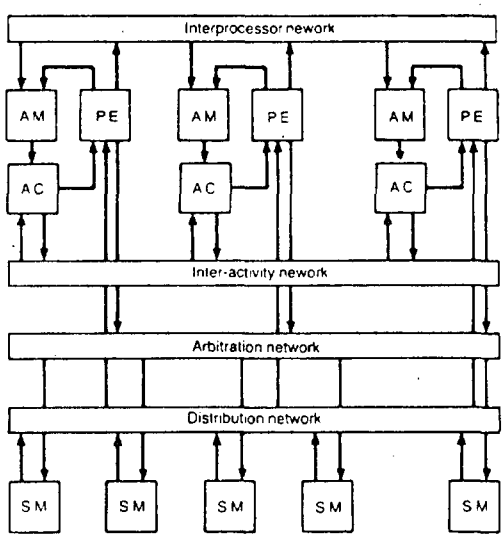


Fig.1-A static dataflow network(delta network).

The dynamic approach gets round the problem of re-entrat code by allowing replication of portions of the network at run time. This has the virtue of simplicity, and may become increasingly feasible as hardware constrains slacken. Fig.2. illustrated one possible configuration using this technique /14/.



A M Activity memory
A C Activity controller
P E Processing element
S M Structure memory module

Fig.2-A dynamic dataflow architecture.

The line of development which holds out the most promise in the short term is the tagged system, variations of which are under development at MIT and Manchester University. Each

data items flowing through the network carries with it an identification tag, which specifies its type (for example it may be a pointer to a large data structure held in fixed store) and its position in the program. The tags enable data items to be paired and matched with appropriate instructions for processing. The tags also indicate the level of recursion if re-entrant code is used. One node of a data-flow systems using this approach /15/ is illustrated in Fig.3.

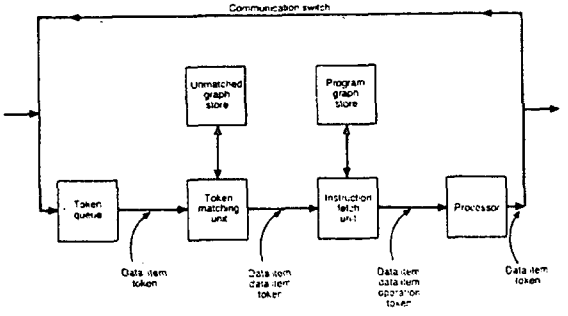
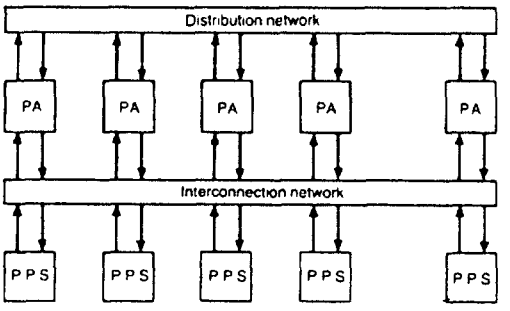


Fig.3-A tagged dataflow architecture.

The graph reduction architecture /16/ is the next variation on the dataflow approach discussed above. This variation is to evaluate functions by working directly on their graphical representations. As various portions of the graph are evaluated, they are replaced by their intermediate results. Evaluation of each of the lowest nodes (which becomes a search to see whether such a node is present in the given relations), can proceed in parallel. The intermediate boolean results are then fed back through the graph as it is reduced, until a single result emerges. ALICE /17/ is the computer which incorporates graph reduction directly into its basic architecture. It is designed to be programmed in the applicative language HOPE, but can also support declarative languages such as PROLOG. The architecture of ALICE enables the parallel operations to be performed without any explicit instructions from the program. Each node in a program graph is represented as a packet within Alice. A packet consists of an identifier fields, a function or operator field, and one or more argument fields, which may be data values or references to other packets. There are also control fields used by the computer in its operation.

The general layout of an ALICE computer is shown in Fig.4. It consists of a large segmented memory serving as a packet pool, and a number of processing agents. The processors and the memory segments are connected by a high-speed switching network which enables any processor to access any memory segment with minimal delay due to other access path. The configuration chosen is a delta network, comprising a large number of simple switching elements with four inputs and four outputs in a regular array. (Fig.1 shows a delta network of elements with two inputs and two outputs.) The network operates asynchronously, so that each request for a packet is propagated through the switches as rapidly as possible, and the packet is returned to the processor as soon as the access path is open.



PA Processing Agent

PPS Packet pool segment

Fig.4-Alice: overall structure.

Also linking each processing agent is a low bandwidth distribution network, which contains addresses of processable packets and empty packets. This network includes simple processing elements which transfer these addresses from one processing agent to another, in order to even out the queue of work waiting at each processor. ALICE uses the INMOS transputer /12,13 / as its basic processing element: each main processing agent cointains a number of transputers, and additional transputers provide the intelligence in the distribution network. The transputer is designed as a single-chip processing element for parallel computer architectures. It has an one-board memory, with high-speed DMA (for input and output channels, bypassing the processor) facilities and reception and transmission registers for data transfer between transputers. Its single sequential processor has a reduced instruction set (RISC processor) for maximum speed (instruction cycle time of 50 nanosecond). Transputer is designed for a very high throughput of data, even if the processing rate is not so high.

The transputer is designed to be programmable directly in Occam programming language /18/. It is intended to be incorporated in a distributed architecture, with individual transputers connected by a very high speed local area network. As such it is an ideal building block for many components of a multi-microprocesor fifth generation computer system.

4. CONCLUSION

With the increased interest in multi-microprocessor and distributed computing systems, there is emerging a large number of proposals and approach to handle them. In a multi-microprocessor design the architectural philosophy requires the interrelated consideration of application requirements, hardware communications, and software aspects. While more detailed treatment of software issues is left until second part which succeed of this paper, each of the considerations, as are: types of communications, priority, co-ordination of processes, process-procesor allocation, network visibility, control issues, and synchronization /19/ should be seen as having implications as to the structure, hardware, and software of a system.

In multi-microprocessor systems the architectural structure, applications requirements, and varied software aspects like the operating systems /20/, communications infrastructure, and tools to aid application programming such as high level languages suitable for parallel programming, all form a tightly knit situation in which it is far more difficult to isolate the constituent parts and arrive at universally accepted solutions.

The requirements of the fifth generation for layers of parallelism and an emphasis on inference rather than numerical computation look like providing sufficient incentive. Even if the objective of a computer with enhanced intelligece is not attained, the new architectures will provide engines of unprecendent power for conventional computing. The move away from general-purpose processors to aggregations of special-purpose chips is likely to affect all branches of information technology. The increase in the scale of integration, and the advanced CAD systems for microchip production will find applications in every branch of microelectronics. The industrial, economic and political consequences of having access to, or not having access to the new generation of silicon foundries are farreaching.

5. REFERENCES

/1/ Bedina M., Distante F. SW & HW Resources Allocation in a Multiprocessor system: An Architectural Problem, Advances in Microprocessing and Microprogramming, EUROMICRO, 1984.

/2/ Tucker A.B., Jr. Programming languages (Second edition), McGraw-Hill Book Company, 1986.

/3/ Stankovič J.A., A Perspective on Distributed Computer Systems, IEEE Trans. on Comp., vol.C-33, NO.12, December 1984.

/4/ Chandy M., Misra J. Distributed simulation: A case study in design and verification of distributed programs, IEEE Trans. on Software Engineering, September 1978.

/5/ Ma P.R., Lee E.Y.S., Tsuchya M. "A task allocation model for distributed computing systems", IEEE Trans. on Comp., pp. 41-47, Jan. 1982.

/6/ Flynn M.J. Directions and Issues in Architecture and Language", Computer IEEE, October 1980.

/7/ Milutinović V. A High Level Language Architecture: Bit-Slice-Based Processor and Associated System Software, microprocessing and Microprogramming 12, 1983.

/8/ Kogge P.M. The Architecture of Pipelined Computers, McGraw-Hill Book Company, 1981.

/9/ Treleaven P. Fifth generation computer architecture analysis, in Moto-Oka, 1982, pp. 265-275.

/10/ Hwang K., Briggs F. Computer Architecture and Parallel Processing (International Student Edition), McGraw-Hill Book Company, 1986.

/11/ Brajak P. Paralelno procesiranje: arhitektura 90-tih godina, Zbornik jugoslovenskog savjetovanja o novima generacijama računala, MIPRO 86, Rijeka, Maj 1986, str.33-46.

/12/ Smith K. New computer breed uses transputers for parallel processing, Electronics 56,4, 1983, pp.67-68.

/13/ Mihovilović B., Mavrič S, Kolbezen P. Transputer-osnovni gradnik večprocesorskih sistemov, Informatica 4/86, Ljubljana, 1986.

14/ Tanaka H., et.al. The preliminary research on the data flow machine and the data base machine as the basic architecture of fifth generation computer systes, in Moto-Oka, 1982.

/15/ Gurd J. Developments in dataflow architecture, in SPL Internacional, 1982.

/16/ Cripps M., Field A.J., Reeve M.J. The design and implementation of Alice: a parallel graph reduction machine, Byte Magazine, June 1985.

/17/ Darlington J., Reeve M.J. Alice: a multiprocessor reduction machine for the parallel evaluation of applicative languages, ACM Confernce on Functional Programming and Computer Architecture, October 1981, pp.65-75.

/18/ Inmos. Occam Programming Manuel, Prentice-Hall, 1984.

/19/ Kolbezen P. Analiza multiprocesorskega sistema, IJS Delovno poročilo Dp-4461, Univerza E.Kardelja, IJS, Ljubljana, Decemaber 1986.

/20/ Kolbezen P. Problemi načrtovanja multiprocesorskega sistema, IJS Delovno poročilo Dp-4462, Univerza E.Kardelja, IJS, Ljubljana, December 1986.

/21/ Foster M.J., Kung H.T. The design of special-purpose VLSI chips, IEEE Computer Magazine 13,1, 1980, pp.26-40.

/22/ Seitz C.L. Conccurent VLSI Architectures IEEE Trans. on Comp., Vol.C-33, No.12, Dec. 1984.

LANGUAGE CONSIDERATIONS OF PARALLEL PROCESSING SYSTEMS
PART TWO: Survey and analysis of concurrent languages

INFORMATICA 2/87

UDK 681.3.06:519.682/.683

Peter Kolbezen
Institut »Jožef Stefan«, Ljubljana

ABSTRACT - The paper is devoted to outlines of Ada and Occam, comparison of communication-oriented features of both languages, and a brief description of the key features of various proposals of some remaining multi-microprocessor oriented languages that have been made in recent years and that the major research issues still remain to be addressed.

O JEZIKIH SISTEMOV PARALELNEGA PROCESIRANJA. DRUGI DEL: Pregled in analiza konkurenčnih jezikov. Članek je posvečen opisu jezikov Ada in Occam ter primerjavi komunikacijsko orientiranih značilnosti obeh jezikov. Podan je tudi kratek opis bistvenih lastnosti različnih predlogov nekaterih ostalih večprocesorsko orientiranih jezikov, ki se pojavljajo v zadnjih letih, in katerih razvojni uspehi se bodo morali še izkazati.

1. INTRODUCTION

Conventional programming languages are procedural: they describe, in exhaustive detail, the steps which a computer must carry out in order to perform a particular data processing task. These programming languages in their present form have no place in the fifth generation of computers, but some recent advances, notably the incorporation of parallelism, and the emergence of the language OCCAM, may mean that the thread of development which started with Algol in 1957 is continued into the computers of the future. A measure of "controlled" parallelism can be incorporated into procedural programming languages. This approach was pioneered in Concurrent PASCAL /1/, Concurrent C /2/, Pascal-Plus, Modula /3,4/ etc. and has been incorporated into the real-time language ADA /5/.

The most fully developed implementation of parallelism into procedural languages is OCCAM, where each module is regarded as a communicating sequential process, which may run on its own, possibly dedicated, processing element. Languages developed to support concurrent processing, allow cooperation between processes but reflect their inheritance both from the "parent" language structure and architectural restrictions. In these languages, called procedure-oriented languages /6/, process interaction is based on shared variables. These languages contain processes (active objects) and modules, monitors (pasive objects) providing means for ensuring mutual exlusion of concurrent accesses. The primitives of these languages do not match the implementation features of the architecture of processor networks. Both message- and operation-oriented languages /6/ are based on message passing but show different views of process interaction. New programming languages like Ada or Occam try to explore structuring concept in concurrent programming. Both languages adapt the process (called task in Ada) as the fundamental notation in program construction but differ significantly in their mechanisms for process comunication and synchronization. A message-oriented language as Occam provides channels and input-/output processes as the primary means for processes interaction while an operation-oriented language as Ada provides remote (extended) procedure call for process interaction. Primitives based on these notations can be implemented on different basses not only by programming languages but by operating systems, special programs, architectures (microprograms) or user programs.

In order to give some details of language development results, the following sections are devoted to outlines of ADA and OCCAM, comparison of communication-oriented Ada and Occam features /7/, and a brief description of the key features of various proposals of some remaining multi-microprocessor languages that have been made in recent years.

2. ADA

Ada has been developed essentially as a general purpose, high level language standard. To have as wide an application scope as possible, constructs have been introduced to help with concurrency.

For real-time aplications, Ada provides facilities for multitasking, i.e. for logically parallel threads of execution that can co-operate in carefully controlled ways.

Tasks A task is an independent thread of execution. Like a package, a task is divided into a specification part and a body. Similarly, the modularity and abstraction concepts for packages generally apply to tasks. The task specification contains entry declarations that define the procedure-like call that can be made to communicate with the task. The task body contains the code and variables (i.e. the internal state) which define the behaviour of the task.

Below is an example of a task to provide an asynchronous buffer between a line-orieted producer and a character-oriented consumer. This task declaration is assumed to be in the context of declarations of the types LINE and CHARACTER; the latter is, in fact, a predefined type.

```
task LINE-TO-CHAR is                                    *
  entry PUT-LINE (L : in LINE);
  entry GET-CHAR (C : out CHARACTER);
end LINE-TO-CHAR;
task body LINE-TO-CHAR is                              **
  BUFFER : LINE; begin
  loop
    accept PUT-LINE(L : in LINE)do                    ***
      BUFFER : = L;
    end PUT-LINE;
    for I in BUFFER'FIRST ..  BUFFER'LAST loop
      accept GET-CHAR (C : out CHARACTER) do
        C : = BUFFER(I);
      end GET-CHAR;
    end loop;
  end loop;
end;
```

```
-----------------------------------------------
  * the task specification
 ** the task body
*** the accept statements and its body
-----------------------------------------------
```

In this example, the task specifications declares the entries PUT-LINE and GET-CHAR for use by other tasks. The body declares a local variable BUFFER which is used to hold a complete line until all of its characters have been transmitted.

The code for the task consists of an unbounded loop with two basic parts. The first part is an accept statements for entry PUT-LINE. The accept statements looks quite similar to a procedure declaration; it has formal parameters and a body. The task waits at this point until some other task calls the entry PUT-LINE to provide data for the buffer. The second part is another loop that transmits one character at a time.

The above example illustrates the the declaration of a single task object. Task types can also be declared, and any number of tasks of that type, i.e. all with the same properties, can be declared as object. Indeed, because tasks are objects, they can be components of records or arrays. They can be pointed to by access objects, passed at parameters, created dynamically, and so on.

Intertask communication When an entry has been called and the called task reaches an accept statement for that entry, a "randezvous" is said to occur and only a single thread of execution is active for the duration of the accept statements. This single thread of execution can be thought of as belonging to either or both of the two task. At the end of the accept statement the rendezvous is complete. Both tasks then continue independently and asynchronously.

In the last example given, when the task is waiting at PUT-LINE, a call on GET-CHAR is queued and the caller must wait. Conversely, when the task is waiting at GET-CHAR, a call to PUT-LINE is queued. Any number of task can be waiting for a given entry to be accepted. In this way, coordination is assured between tasks that provide lines and those that consume characters.

While the last example did not require more than one accept statement, there can be any number of accept statements for each declared entry. Other features allow accept statements to be executed conditionally.

A select statement allows a task to wait until a call is received on any one of a set of entries. If more than one call is waiting when the select starts, one of them will be chosen arbitrarily for processing.

The set of possibilities need not be the same on every execution of the select statement. An optional "when clause" evaluates the "true" condition for the entry to be "open", i.e. eligible to be accepted. If none of the alternatives are open and the optional else alternative appears in the select statements, it will be chosen. An exception is raised if no altrnative is open and no "else" alternative is provided.

Other variations of the select statements include a delay or terminate alternative which can appear in place of an accept statements. However, neither a delay nor terminate alternative can be used in combination with the "else" alternative. If one of the alternatives of the select is a delay and there are no calls waiting for any of the open alternatives, the task will wait no longer than the specified amount of time before preceding. A delay value of zero means that if no entry call is immediately available, the task will not wait at all.

A terminate alternative is a particularly intersting construct that adresses the questions, "How do you gracefully shut down a set of cooperating tasks without running the risk of error?" When a task is waiting at a terminate alternative, it effectively declares to the run-time environment that is prepared to be terminated provided that all other task which it can interact directly or indirectly are also at terminate alternatives or have completed. When all such tasks are in this state, none of the tasks of the set will ever receive an entry call (they are all waiting for some other task to make an entry call), so it is safe to terminate them all. Of course, if an entry call is received by a task that is prepared to terminate, the "okay-to-terminate- me" status is rescinded and normal processing resumes.

The language also supports a conditional entry call in which the calling task is not suspended and the entry call is not made unless the called task is ready to accept it immediately. There is also a timed entry call-a generalized form of the conditional entry call-in which the call is cancelled if it is not accepted within a specified amount of time. These two foras of entry call make it possible for a calling task to assure that it will not be indefinitely delayed by any task that it calls.

Priorities and scheduling A task can either have a priority or not. A priority is specified by a pragma. If a priority is given, it must be a compile-time static value and cannot be changed during execution. When no priority is given, the compiler is free to choose when scheduling decisions need to be made, when the overhead of the scheduling decision can be

avoided, and at what priority the task is run when it starts executing. The overhead of tasking in Ada is small. When implemented on a single procesor, it is comparable to the overhead of procedure calls.

Interrupts External devices are treated in Ada as tasks, and interrupts are treated as entry calls. The priority of the external device is *dependent on its hardware priority and* is guaranteed to be higher than any software priority. In this way, interrupts can be directly "connected" to accept statements. A consequence is that the tasking model is powerful enough to incorporate the conventional view of interrupts and interrupt handlers.

ADA has been widely criticised for its size and the awkwardness of its constructions. Although it is designed to be amenable to formal methods of proving the correctness of program modules, much work remains to be done before this is possible on a wide scale. The unease about ADA stems from the fact that an error in an ADA system could have catastrophic consequences. Nevertheless the Department of Defense continues to insist on the use of ADA for all future real-time applications. A small number of Ada *program support environments are under development*, and the experience gained on these will undoubtedly be of benefit to fifth generation software engineering work.

3. OCCAM

OCCAM is a programming language designed to support concurrent processing, in computer systems where many processing elements operate independently but interact (Inmos, 1984). It is the low-level language of the Inmos transputer. Like ADA, it can be used for real-time embedded systems, but it is much simpler - it is named after the medieval philosopher who pioneered the idea of Occam's razor, a sharp intellectual instrument used to cut away all superfluous details in a system.

Occam is directly based on the idea of communicating sequential processes (CSP) /8/ and its definition and implementation were advised by C.A.R.Hoare, 1978, the designer of CSP. A unit of concurrency is called a process in Occam, and a task in Ada. A process performs a sequence of actions, and may communicate with other processes via defined channels. Communication along a channel is synchronous: when both an input and an output process are ready to communicate on the same channel, the data item is transferred, and the processes is illustrated in Fig.5. Data types in Occam are restricted to integers, characters, arrays and strings.

Fig.5-Hardware configuration for
Occam program example.

Occam process definition

```
declaration::=PROC ident.  [form.params]=process

 VAR identifier [[ ]] { ,identifier [[ ]] }
    --variable(vector of variables)
CHAN identifier [[ ]] { ,identifier [[ ]] }
    --channel(vector of chanels)
VALUE identifier [[ ]] { ,identifier [[ ]] }
    --value(vector of values)

process::=declaration
          process
```

Interprocess communication As an example, consider two simple hardware devices, one of which merges signals from three incoming channels, and the second of which buffers the output from the first, before passing it down an outgoing channel. See Fig.5. An Occam program to control this configuration is as follows:

```
chan link:
par
   while true
      var x:
      alt
          c1 ? x
             link ! x
          c2 ? x
             link ! x
          c3 ? x
             link ! x
   while true
      var x:
      seq
          link ? x
          c4 ! x
```

In Occam a channel declaration CHAN introduces a new identifier to be used as a channel. A channel may be a simple channel or a element of a vector of channels. Channels are used to communicate between concurrent processes. Direct (and static) channel naming is used and message passing is synchronous. In the above example the two "while true" constructions operate in parallel as endless loops. Within the first construction, the "alt" keyword chooses whichever of the three channels is ready for input. The input constructions (c1 ? x, meaning input the value of x from chanel c1, etc.) guard their corresponding outputs (link ! x, meaning output the value of x on the channel named link). The second process repeats indefinitely the sequence of input from the link channel and output to c4. Note that the structure of the program is created almost entirely by indentation: the scope of the par, while, alt and seq constructors is the indented lines below them.

The virtue of Occam is its simplicity. It has fewer than thirty reserved words, and only a small number of constructors. Although each process uses destructive assignments, the use of channels for inter-process communication makes it entirely consistent with dataflow and graph reduction computer architectures. Occam was designed with computer architectures of this nature in mind, and with a view to fifth

Fig.5.a - Communicating sequential processes

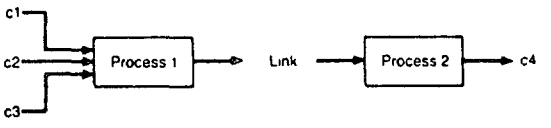


Fig.5.b - Hardware configuration for
Occam program example

generation applications. It is in tune with the fifth generation philosophy of developing the hardware and sofware of a computer system together. Together with the Inmos transputer, it provides a modular hardware/software component of the type which is essential in the construction of highly parallel computer systems. The simplicity of Occam makes it an appealing prospect for proving the correctness of processes. Its lack of data structuring facilities, and closeness to the hardware of the computer system mean that Occam is likely to be the low-level language of fifth generation of systems, with applications written in a more abstract language. This is the case with the Alice computer.

4. COMPARISON OF ADA AND OCCAM FEATURES

Features of communication languages are: Unit of concurrency, specification, means of communications, correspondence, naming (identifications), service and library programs, identity checking, synchronization, and nondeterminism.

A unit of concurrency is called a process in Occam, and a task in Ada.

A specification unit of concurrency is called a process in Occam, and a task in Ada.

Means of communication are channels and input-/output processes in Occam solution and reaote procedure call (rendezvous) in Ada solution.

A correspondence between the calling and the called programs (processes) determines a static one-to-one correspondence in Occam solution while in Ada solution there are multiple calls and queueing.

In Ada solution there is an asymetric naming (identification) between calling and called tasks. However, a naming is symetric in Occam solution.

The asymetric solution of naming enables the called module to be written as a library module (server) without knowledge of the names of its callers. So a service and liberary programs are facilitates in Ada solution and are not available in Occam solution.

A identity checking is simple in Occam and is complicated in Ada.

Both Occam and Ada can perform synchronization (without message passing). It is accomplished in Ada by parameterless entry calls. If the ANY keyword is used instead of a variable in an input or an output transfer in Occam, the input value is discarded or an arbitrary value is the output. The solutions of Ada and Occam are mostly equivalent.

In Occam guarded alternative processes are used to select one of the processesto be executed. The expression evaluation of the guards specifies the selections. An alternative process waits until at least one guarded process is ready for executions. The selection is arbitrary and therefore this is a means of expressing nondeterminism. Ada's statements, in both forms: select (else) statements and conditional entry calls, enables nondeterministic waiting for communication with other tasks.

5. THE ANOTHER PROPOSALS OF MULTI-MICRPROCESSOR LANGUAGES

CSP

Distributed program /8/ is defined as processes, with mutually disjoint address spaces, communicating directly using typed messages. Arrays of processes can be defined.

Send and receive of messages are termed as input and output operations, respectively. Receiver and sender are to be synchronized for message transfer.

Non-determinism is permitted through guarded commands. Input statements can appear in guards. A parallel command permits the simultaneous initiation and execution of several processes.

PARLANCE

A distributed program /9/ is obtained by processes communicating using messages through typed ports.

Hierarchies of processes can be defined, but only the processes of the lowest levels can have executable codes.

The minding of ports is done external to the process definitions. Each port of a process is connected to only one port or some other process. The bindings are static. Sender and receiver are to be synchronized for message transfer (input-output).
A parallel command which involves only input and/or output statements is permitted (i.e.AND-WAIT on multiple ports: all message transfers, in any order, must be completed before the command execution is deemed to be completed).

Non-determinism cannot be handled.

TASK

TASK /10/ is an extension of BLISS and is based on the "object model" supported by the StarOS operating system for Cm*.

Different object types that exist are: task force, module, process, basic, stack, deque, device, mailbox.

A TASK specification affects only the creation and initialization of objects.

One codes algorithms in BLISS and specifies logical structures in TASK.

The language implements both procedure invocations and message-passing capabilities. A procedure invoked by a process could be executed in parallel with the execution of the caller. Communication is through mailboxes.

Dynamic definition and the creation of task forces are possible.

NSL

NSL /11/ is a high level definition language used for specifying the detailed requirements on any particular layers in the ISO model of a distributed system, i.e. a language for the specification of protocols.

Processes communicate through typed messagees. The dynamic creation of processes at a desired

phisical site/note of the network is possible. The destruction of a process by its creator is permitted, which results in the destruction of all its descendent processes.

Network data objects, like files, can be shared.

Certain protection mechanisms are provided.

PCL

Distributed program /12/ components are processes, memory segments, ports, and links. Components can be in five states (uninstantiated, bound, expanded, initialized, and executing), and can be shared.

Ports are connected by links. Various types of links can be defined (including some performance attributes). The presence of a message at a port can be tested.

Highly dynamic process configurations can be instantiated during executing, using a static description of the process structure. Hierarchical structuring of processes (called clusters) is also possible. Process structure descriptions can be given in terms of static or structural characteristics.

Distributed Processes /13,14,15/ Processes contain local data, initialization statements, common procedures, which can be called by other processes. Those calls are termed external requests.

Unsymmetric communication is implemented by procedure call semantics for message transmission.

The initial statements and external requests of a process are executed serially. They terminate or wait for some condition to become true. The execution of a process is similar to that of a monitor.

Guarded commands and guarded regions provide non-determinism. The latter can delay the execution of an operation.

An array of processes is possible.

Synchronizing Resources /16,15/ A distributed program consists of a set of resources, each of which encapsulates a set of processes and shared variables.

Processes contain declarations of externally callable operations (which are similar to the procedures of distributed processes and entires of Ada), which can have associated priorities.

Intra-resource process communication is done through operations on shared variables.

Inter-resource process communication is implemented through operations.

A multiprocess is a multiple number of those operations defined in a multiprocess which can be executed simultaneously.

Extended CLU.

A distributed program /17/ is achieved by one or more "guardians".

A guardian is a number of processes plus objects. Each guardian is expected to provide synchronization, security, backup, and recovery for the resources it protects. Each guardian exists entirely in a node of the underlying physical network.

Object are passive; direct manipulation by processes is possible only through the invocation of predefined operations.

The dynamic creation of guardians is possible, but they are immovable from the nodes where they are created.

Inter-guardian communication exist only through messages. Time-outs are permitted in message reception statements, so that some exeption conditions can be handled.

Intra-guardian communication is achieved through shared objects.

Buffered messages are sent to typed ports. Compile time type checking can be quite comprehensive.

MOD

MOD /18/ is drived from MODULA /14/. Separate compilation is posible.

Processor modules contain processes and message types used by those processes. Variables can be shared by processes in a processor module, similar to the guardians in Extended CLU.

The network module links processor modules, and declares global types and constants.

The interface module is similar to monitor: mutual exclusion across all the contained procedures.

A process is initiated for every message.

Processes have dynamically variable priorities associated with them.

Extended POP-2

Extended POP-2 /19/ is, as the name implies, an extension of the artifical intelligece language POP-2.

Parameterized process types with input and/or output ports can be defined. Parallel processes (with connecting ports, either new and/or old ones) can be created dynamically, thus facilitating shrinking or expansion of process hierarchies.

Recursion is permitted, but non-determinism cannot be handled.

Ports are connected by channels. Each channel can have multiple readers (with only one writer) and all of them receive all the messages sent along the channel.

Actors /20/ A distributed program consists of dynamically creditable actors (which are somewhat like processes), communicating through buffered messages. Messages may transmit actor's names.

Non-determinism, recursion, and time-outs can be specified.

Serializers are used to control access to protected resources. The serializer mechanism is a generalized and improved monitor mechanism.

PLITS/ZENO

A distributed program /21/ is obtained by multiple modules (which are similar to processes) communicating solely through buffered messages.

Modules can be created dynamically, but can not be destroyed by other processes.

All components/fields of a message may not be visible to the recipient of that message. Module names, as well as transaction names, could be used for controlling message receptions and dispatches. Transactions names could be considered as dynamically created mailboxes.

PLITS/ZENO parts are intentionally not strongly typed.

FLOWGRAPH

A flowgraph /22/ is only a model, not a language.

Processes communicate through typed ports of channels, with unbufferd messages.

Recursion and non-determinism can be expressed.

Processes and channels can be created dynamically.

LIMP

A distributed program /23/ consists of processes communicating solely through unbuffered messages.

Processes can be created dynamically.

Typed channels (possibly with multiple readers and writers) link processes for message transfers. The read capability for a channel is called a "source" and the write capability a "sink".

Channels, sources and sinks can be created dynamically and passed around in messages.

CCS

CCS language /24/ is specialised for synthesis tools. This report also outlines a synthessis tools. CCS may define several parallel processes. They perform action by transporting messages between each other. Because there are no means to describe time delays CCS cannot fully describe a behaviour. It may only be used to describe event synchronization.

This language can describe structures of parallel processes. A process is described by a sequence of statements describing by effect the actions of a process (imperative parallelism).

CCS can also describe parallel processes. Each process is describe in a functional approach (applicative parallelism).

True hardware consists of (hierarchical) structures of processes, each acting dependent on values of other processes. Such parallelism can in a straight forward manner be modelled by Ada and CCS (or Simula).

On a high-level in a system components generally communicate by sending messages. On this level CCS and Ada can model behavior. Both languages synchronise in this way.

ELLA

The ELLA language /25/ is an applicative language for RTL level behaviour. It can describe structures of parallel processes (imperative parallelism) and can also describe parallel processes (applicative parallelism)

6. A NON-PROCEDURAL LANGUAGES

Fifth generation programming languages will have to cope with highly parallel architectures, with different types of processing elements for knowledge base management, inference processing and intelligent intrface support, and with applications based on intelligent knowledge-based systems. They will need to be in tune with the philosophy of software engineering, with its requirements for cost-effective, and software development on integrated programming support environments. Furthermore, fifth generation programming languages require a firm base in the concepts of artificial intelligence, which underlie all aspects of fifth generation computer systems. The general line of development in programming languages which takes account of these requirements is the range of languages which may be termed non-procedural. Such languages are based on a description of what is required of a processing task, and the structure of the data required for the task, but leave to the language processor the details of how the task is to performed. Many non-procedural languages are based on the principle of non-destructive assignments: the value of a particular data item, once set, is not altered during the running of the program module in which it occurs. This property is significant if the computer architecture is not based on static memory, but on data flowing from one store to another.

In mainly exists two types of non-procedural languages: declarative languages and applicative languages.

LISP has been the mainstream artificial intelligence programming languages for many years. It has been the starting point for a number of more recent declarative programming language, notable PROLOG. Recent work on language development includes PARLOG, a parallel implementation of PROLOG, and various attempts to synthesise elements of PROLOG and LISP, or PROLOG and procedural language /26/. The Icot programming language development plan is for a sequential logic-based language, KL0, followed by a parallel version, KL1 and subsequent versions to match the hardware evolution of fifth generation systems.

Although the "upper" layers of software of a fifth generation computer are likely to be declarative in nature, at lower levels there is a need for a procedural language which can operate on large, complex data structures. LISP /27/, or a derivative of it, is an ideal candidate for this role. One possible evolution of LISP is into an applicative language, with no destructive assignments, which will bring it into line with dataflow and graph reduction architecture.

Applicative languages are a class programming languages which pre-date fifth generation computers, but which may be drawn into the mainstream of the new generation. They arose in the evolution of ideas of program structuring, and the need for formal procedures of program design and verification. Programs in this

language consist entirely of functions; there are no procedures. Assignments are non-destructive. The statements in an applicative language are not to be interpreted in sequence. One consequence of this is that there are no loop constructions in an applicative language.

These properties make applicative languages suitable for fifth generation computer development (data flow, graph reduction or other parallel supporting hardware architectures). An example of an applicative programming language is HOPE, adopted for the ALICE computer.

7. CONCLUSION

The increasing use of microelectronic components for dedicating applications has resulted in the need for high-level design tools to assist in the design process. Suitable languages must be chosen for both the behavioural and structural descriptions which allow the concise translation of one to another; viz, compilation in the programming sense. This approach would permit an "optimised" structure to be synthesised for a particular application.

The behavioral description high-level programming languages are, for example, OCCAM and ADA. In these cases there is, generally a one-to-one mapping of the language constructs into hardware primitives or the language semantics have been altered to alow simpler hardware translations. The behavioural description language OCCAM allows a level of process abstraction, a description of concurrecy and the formulation of an implementation independent definition. Whilst not the ideal language, OCCAM does permit behaviour to be expressed in an elegant, concise manner and has the added advantage that behaviour can be simulated at an early stage in the design cycle. The structural description language ELLA provides mechanisms for defining a component at both the functional and structural levels in a hierarchical manner, together with accompanying simulation facilities.

The main characteristics of programming languages outlined in this chapter is that they are very different from their immediate predecessor. They are non-procedural, non-sequential and non-numeric in emphasis. They will require different programming skills from those practised today, a point with profound consequences for a world-class IT industry in the next decade.

At present it is not clear whether language will become the base language of fifth generation. This paper attempts to give survey over characteristics of any more known concurrent languages and to make the point that there is a wide choice in the matter, and that a number of languages might be used for the various layers of software controlling the processing assemblies which make up a multi-microprocessor (fifth generation) computer system.

8. REFERENCES

/1/ Milutinović V. Tutorial on Advanced Microprocessors and High-Level Language Computer Architectur, IEEE Comp. Soc. Press, 1986.

/2/ Tsujino Y., et al. Concurrent C: A Programming Language for Distributed Multiprocessor systems, Software-Practice and Experience, 14, 1984, pp. 1061-1078.

/3/ Tucker A.B.,Jr. Programming languages (Second edition), McGraw-Hill Book Company, 1985.

/4/ Andrews G.R., Schneider F.B. Concepts and Notations for Concurrent Programming, ACM Computing Surveys, 15, 1985, pp.3-43.

/5/ Young S.J. An Introduction to Ada, Ellis Horwood, 1984.

/6/ Vajda F. Concurrent Systems, Programming Primitives and Languages: A Comparative Study, Microprocessing and Microprogramming 18, 1986, pp.185-194.

/7/ Ramming F., Chair. High Level Systems: Simulation, Specification & Synthesis, Microprocessing and Microprogramming 18, C2, 1986.

/8/ Hoare C.A.R. Communicating sequential processes, CACM, August 1978.

/9/ Hansen P.B. Distributed process: a concurrent programming concept, Communications of the ACM, 21, No 11, Nov. 1978.

/10/ Jones A, Schwans K. TASK forces: distributed software for solving problems of substantial size, Proc. IV Int. Conf. on Software Engineering, September 1979.

/11/ Tarini F., Sharp R.,Martelli M., Endrizzi A. A network system language, Proc. I Int. Conf. on Distributed Computing Systems, October 1979.

/12/ Lesser V., Serrain D., Bonar J. PCL: A process-oriented job control language, Proc. I Int. Conf. on Distributed Computing Systems, October 1979.

/13/ Hansen P.B. Distributed process: a concurrent programming language, CACM, November 1979.

/14/ Stankovic J.A. A Perspective on Distributed Computer Systems, IEEE Trans. on Software Engineering, September 1979.

/15/ Kolbezen P., Multiprocesorski operacijski sistemi (problematika načrtovanja, značilnosti in razvojni doseţki), IJS Delovno poročilo Dp-4463, Univerza E.Kardelja, IJS, Ljubljana, Marec 1986.

/16/ Haare C.A.R. Synhronization of parallel processes, in Book and Advanced Techniques for Microprocessor Systems, Hanna F.K., ed., Peter Perigrinus, London, 1980.

/17/ Liskov B. Priaitives for distributed computing, Proc. VII Symp. on Operating Systems Principles, December 1979.

/18/ Cook R. MOD-Alanguage for distributed programming, Proc. of the I Int. Conf. on Distributed Computing DSystems, October 1979.

/19/ Kahn G.,MacQueen D. Coroutines and networks of communicating processed, Proc. of the I Int. Conf. on Distributed Computing Systems, October 1979.

/20/ Hewitt C., Attardi G., Liberman H. Security and modularity in message passing, Proc. of the I Int. Conf. on Distributed Computing Systems, October 1979.

/21/ Feldman J. High level programming for distributed computing, CACM, June 1979.

/22/ Milne G., Milner R. Concurrent processes and their syntax, JACM, April 1979.

/23/ Carlstedt G. A language for behavior, structure and geometry, Microprocessing and Microprogramming 18, 1986, pp.567-580.

/24/ The ELLA language reference manual, Praxis Systems Ltd, England, 1985

/25/ Campbell J. (ed.) Implementations of Prolog, Ellis Horwood, 1984.

/26/ De Saram H. Programming in micro-Prolog, Ellis Horwood, 1985.

/27/ Ennals R., Beginning micro-Prolog, Ellis Horwood, 1983.

/28/ Hoare C.A.R. Communicating sequential processes, Communications of the ACM 21,8, 1978, pp.71-92.

MODEL INFORMACIJSKEGA SISTEMA ZA PODPORO KADROVSKI DEJAVNOSTI

INFORMATICA 2/87

UDK 658.3:681.3

Ivo Ščavničar
Železarna Jesenice

Prispevek podaja opis osnutka informacijskega sistema za podporo kadrovski dejavnosti, oblikovanega v železarni Jesenice. Opis predstavi podatkovni model predlaganega sistema, drevesno razgradnjo funkcij kadrovske dejavnosti, ki jih bo podprl predlagani sistem in opredelitev vloge elementov organizacijske strukture. Opuščena so ostala vprašanja, ki sicer sodijo v osnutek informacijskega sistema: potrebna strojna in programska oprema, predvideni stroški, podrobnejša opredelitev postopkov, izgled in vsebina nosilcev podatkov.

INFORMATION SYSTEM MODEL FOR PERSONNEL EVIDENCE ADMINISTRATION APPLICATION. The paper describes conceptual design of information system intended as support for personnel evidence administration, developed in Železarna Jesenice. Description presents data model of suggested system,tree decomposition of functions involved in personnel administration which are supported in designed system and defines the role which elements of organization scheme do have in designed system. Other questions, which otherwise take part in design of information system such as: demanded HW and SW support, expected costs, description of processes, content and appearance of individual informations, are not discussed.

0. UVOD

Računalniško podprti informacijski sistemi s področja kadrovske dejavnosti so gotovo med tistimi, ki jih srečamo najpogosteje. Zato v opisih teh sistemov srečamo marsikaj znanega. Kljub temu je tak opis lahko zanimiv, če gre za sistem oblikovan v zadnjem času, saj razvoj na področju informatike prinaša nove možnosti, iz dneva v dan pomembnejša vloga človeka na vseh področjih dejavnosti pa nove zahteve.

Kadrovska dejavnost spada med pomembnejše v železarni Jesenice, saj sodimo med večje kolektive (nekaj nad 6000 zaposlenih), izobrazbena struktura je raznolika, dokaj živahno menjamo in posodabljamo tehnologijo ob tem pa se večkrat otepamo s pomankanjem ustreznih kadrov. Obstoječe računalniške obdelave s tega področja oblikovane pred mnogimi leti so namenjene predvsem izračunavanju osebnih dohodkov, podpore širši kadrovski dejavnosti pa nudijo le malo. Zato smo se že v letu '81 odločili, da uvedemo nov informacijski sistem, ki bo v večji meri kot obstoječi ustrezal željam in potrebam uporabnikov.

Sistem, ki ga pričakujejo uporabniki naj bi avtomatiziral številna rutinska opravila, omogočal različne analize pomembne za odločanje na področju kadrovkse politike, poenostavil oblikovanje poročil, ki jih zahtevajo različne ustanove, predvsem pa omogočal čim učinkovitejše razporejanje delavcev na delavna mesta. Da bi zagotovili kar največjo ažurnost podatkov in trenutno dosegljivost vsaj najpomembnejših informacij želijo uporabniki sistem, v katerem se bo čim več postopkov izvajalo interaktivno.

Od že izdelanih paketov, ki jih je mogoče kupiti smo si ogledali INTERPERS, izdelek podjetja IBM, ki pa ni v celoti ustrezal našim zahtevam. Zato smo se odločili za samostojno izdelavo ustreznih programov. Po začetnih aktivnostih je (zaradi nalog z višjo prednostjo in organizacijskih sprememb na nivoju delavne organizacije) delo zastalo in se nadaljevalo šele v letu '86. Zaključene so vse aktivnosti sistemske analize razen podrobnih opisov za večji del programov, dober del le-teh pa je že kodiran in na voljo uporabniku.

Cilj članka je predstaviti nekatere rezultate prve faze oblikovanja novega sistema: podatkovni model, funkcionalni model ter vlogo posameznih organizacijskih enot v predlaganem modelu informacijskega sistema.

Vsebina je razdeljena v štiri poglavja. Prvo poglavje je namenjeno opisu podatkovnega modela. Jedro tega opisa predstavlja kanonska shema (mehurčni diagram) vseh relacij, ki v modelu nastopajo. Drugo poglavje predstavi funkcionalno plat predlaganega modela. Poleg sheme, ki prikazuje drevesno razgradnjo funkcij zajetih v sistem, so pomembnejše med njimi tudi kratko opisane. Tretje poglavje prikazuje vlogo elementov organizacijske sheme v predlaganem modelu. Poleg sheme drevesne organiziranosti sta v poglavju podani tudi matriki, ki prikazujeta vlogo vsakega od organizacijskih elementov pri vpisovanju in uporabi podatkov (iz podatkovnega modela) oziroma odgovornost za izvajanje posameznih funkcij (iz funkcionalnega modela). Zaključek poskuša predlagani sistem primerjati z obstoječim in oceniti prednosti, ki jih prinaša.

Snov je podana le v osnovnih obrisih, v podrobnosti se ne spušča. Slednje so v preveliki meri odvisne od konkretnih pogojev in potreb v žele-

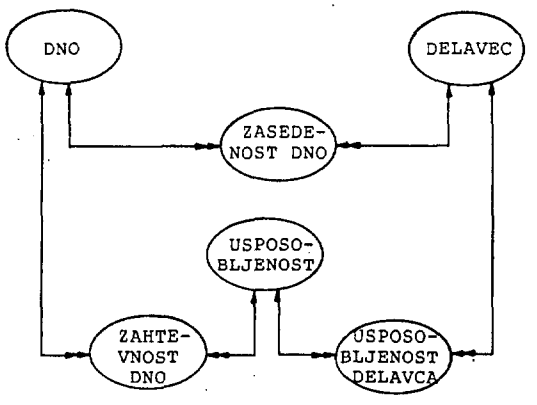
zarni Jesenice in zato nezanimive za ostale. Opis predlaganega modela in v njem nakazane rešitve pa bodo kljub temu marsikomu, ki se ukvarja z informatiko na področju kadrovske dejavnosti, ponudile vsaj kakšen koristen namig če že ne primer vreden vsaj delnega posnemanja.

1. PODATKOVNI MODEL

Podatkovni model predlaganega sistema je relacijski in vse relacije v njem so v tretji normalni obliki v smislu definicij iz (1), (2), (3) ali (7). Jedro opisa predstavlja kanonska shema, v drugem delu pa so kratko opisane relacije. Kanonska shema je podana grafično v obliki mehurčnega diagrama, kot je predlagano v (1), le da je naša shema nekoliko poenostavljena. Mehurčki predstavljajo relacije ne pa, kot je to predvideno v (1), posameznih tipov podatkovnih elementov (atributov). Črte, ki povezujejo mehurčke, prikazujejo povezave med posameznimi relacijami oziroma obstoj in "cilj" tujih ključev v posameznih relacijah. Podrobneje: črta, ki povezuje relacijo A z relacijo B in je na strani relacije A opremljena z enojno, na strani relacije B pa z dvojno puščico pove, da za vsak element relacije B (zapis, vrstico v relaciji B) obstaja natanko en element relacije A. Za poljuben element iz relacije A obstaja eden, več ali sploh nobeden element iz relacije B. V primeru take odvisnosti med relacijama A in B je eden od ključev relacije A (navadno primarni ključ) hkrati tudi eden od atributov relacije B.

1.1 Kanonska shema

Bistvo podatkovnega modela je zajeto v treh entitetah oziroma njim ustreznih relacijah in odnosih, ki veljajo med njimi. Te entitete so: delavec, delovna naloga ali opravilo in usposobljenost (opisi so podani v razdelku 1.2). Odnos med njimi grafično podaja spodnji mehurčni diagram:



Slika 1: Podshema mehurčnega diagrama

Poleg relacij, ki ustrezajo entitetam delavec, DNO (delavne naloge in opravila) in usposobljenost, v diagramu nastopajo tudi relacije, ki omenjene paroma povezujejo med seboj: zasedenost DNO, ki vsebuje informacijo o razporeditvi delavcev na DNO, zahtevnost DNO, ki pove katere usposobljenosti zahteva posamezna DNO in usposobljenost delavca, in za vsakega delavca pove, katere sposobnosti ima. Tak model naj bi omogočal učinkovito izvajanje osrednjih funkcij kadrovske dejavnosti: analizo potreb, analizo možnosti (razpoložljivih kadrov) in razporejanje delavcev na delavne naloge in opravila. Informacije potrebne za izvajanje teh funkcij so, nekoliko poenostavljeno, vsebovane prav v "trikotniku" entitet delavec, DNO in usposobljenost ter entitetah, ki jih povezujejo.

Gornja shema predstavlja le poenostavljen izvleček iz celotne sheme, ki je prikazana s sliko 2. V razširjenem modelu je relacija usposobljenost zamenjana z več manj splošnimi relacijami: psihološki testi, fiziološki testi in znanje. Temu ustrezno so dodane tudi relacije, ki povezujejo posamezno vrsto usposobljenosti z relacijami DNO in delavec.

Celoten model je seveda bistveno obsežnejši, saj kadrovska dejavnost poleg osnovnih funkcij (analiza možnosti in potreb ter razporejanje) izvaja tudi mnogo drugih, ne dosti manj pomembnih. Mehurčni diagram celotnega modela je prikazan na sliki 2.

Za lažje razumevanje diagrama in opisov relacij še opomba:
Relacije v podatkovnem modelu predstavljajo bodisi katerega od tipov entitet pomembnih za kadrovsko dejavnost, bodisi povezavo, ki obstaja med entitetami. Posebnost so povezave med entitetami istega tipa. Značilen primer take povezave v modelih, v katerih nastopa entiteta tipa izdelek, so "sestavnice" (izdelki so enostavni ali sestavljeni iz drugih izdelkov). Tudi v primeru predstavljenega modela je več takih tipov entitet oziroma njim ustrezajočih relacij: mentor (entiteta tipa delavec) spremlja delo pripravnikov (tudi entitete tipa delavec), delovna naloga je enostavna ali sestavljena iz več opravil, več testov sposobnosti se združuje v baterije testov ipd. Relacija v kateri obstajajo notranje povezave o katerih bi radi shranjevali infromacijo, zahteva v podatkovnem modelu dodatno relacijo. Ključ slednje dobimo tako, da sestavimo (sklopimo) ključ relacije, katere notranjo povezavo podaja s samim sabo. Za relacije, ki podajajo notranje povezave neke druge relacije, bomo v opisih uporabljali izraz struktura, ki se je udomačil v našem računskem centru.

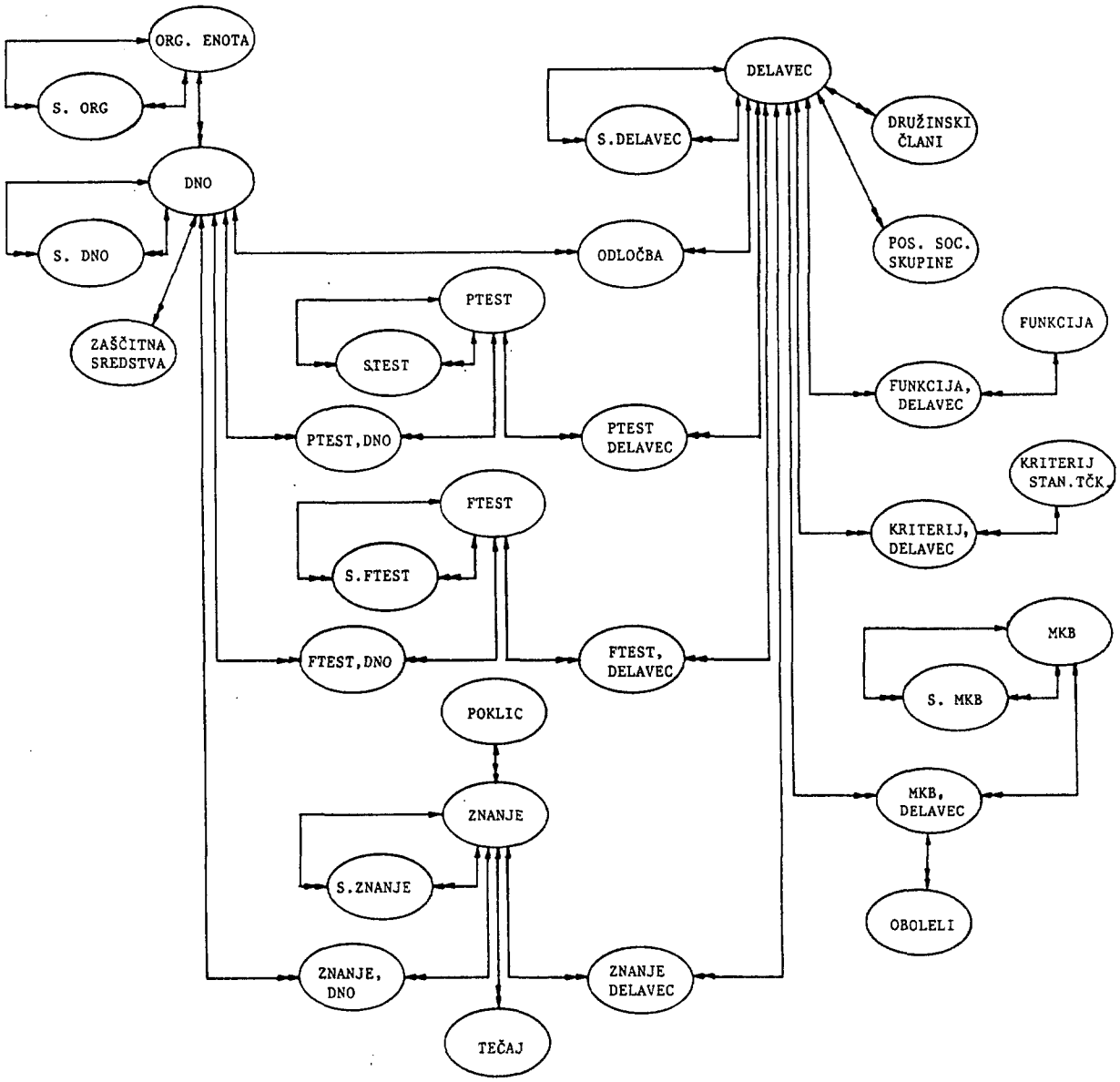
1.2 Opis relacij

Čeprav so že sama imena relacij dovolj zgovorna in njihove medsebojne povezave razvidne iz sheme, za boljše razumevanje modela dodajamo kratke opise posameznih relacij. V opisih so v oklepajih, ki slede imenom relacij, našteti najpomembnejši podatkovni elementi. Podčrtani so tisti, ki sestavljajo identifikator (ključ).

DELAVEC (delavska številka, ime in priimek, rojstni podatki, spol, enotna matična številka, naslov, narodnost,...) je relacija, ki vsebuje podatke o osebah tako ali drugače pomembnih za kadrovsko dejavnost v ŽJ. Ne vsebuje le zaposlenih ampak tudi štipendiste, upokojence, vse, ki kandidirajo za sprejem v ŽJ, vse ki so opravili sprejemni postopek itd.

S.DELAVEC (del.številka,del.številka,vrsta odnosa, datum,...) je relacija, ki omogoča shranjevanje informacij o različnih povezavah med delavci kot je na primer povezava mentor - pripravnik (pomembno za spremljanje izvajanja pripravništva) ali sodniški odnos (pomembno za uveljavljane določenih pravic delavcev). Predstavlja torej strukturo za entitete vrste "delavec". Zapis v tej relaciji lahko predstavlja tudi začasno nalogo (projekt).

DNO (šifra DNO, naziv, nadrejen org.element, normativ števila zaposlenih,plačilna grupa,..) je relacija, ki shranjuje podatke o delovnih nalogah in opravilih. Tudi ta relacija je



SLIKA 2: Mehurčni diagram

strukturirana:delavna naloga je bodisi enostavna (je opravilo) ali sestavljena iz več opravil. Delokrog, ki tudi spada v to vrsto entitet, pa je sestavljen iz več nalog.

S.DNO (šifra DNO, šifra DNO, delež opravila v nalogi,...) je relacija, ki podaja strukturo relacije DNO.

ODLOČBA (delavska številka,šifra DNO,datum odločbe,številka odločbe predvideno število ur na mesec, dejansko število ur v obračunskem mesecu, predviden datum zaključka odločbe, dejanski datum zaključka,...)
Ta relacija nosi podatke o razporeditvi delavcev na posamezne DNO. Vsak (aktivni) delavec ima vsaj en zapis v tej relaciji,lahko pa tudi več. To pomeni,
. da bomo shranjevali tudi podatke o preteklih razporeditvah delavca
. da je delavec lahko razporjen na več nalog hkrati, z različnim predvidenim številom ur za posamezno nalogo.

ORG.ENOTA (šifra org.enote,tip,nivo,naziv enote,...) vsebuje podatke o elementih organizacijske sheme v ŽJ.

S.ORG (šifra org.enote,šifra (nadrejene)org. enote,...) podaja organizacijsko shemo ŽJ. Vključitev te relacije v podatkovni model med drugim omogoča izvajanje različnih obdelav oziroma pridobivanje informacij za poljuben organizacijski element (obrat, delavno skupino,TOZD, DS itd) in združevanje (agregiranje) podatkov po različnih nivojih.

PTEST (šifra testa,naziv,sred. vrednost, mejne vrednosti,...) vsebuje podatke o psiholoških testih, ki jih ob sprejemu in premeščanjih izvajamo v ŽJ.

S.PTEST (šifra testa,šifra testa, vrsta povezave,...) podaja strukturo relacije PTEST. Poleg informacije o združevanju testov v baterije vsebuje ta relacija tudi informacijo o možnosti zamenjave (ekvivalence) med testi.

FTEST
analogno kot PTEST za fiziološke in zdravstvene teste.

S.FTEST
podaja strukturo fizioloških testov.

ZNANJE (šifra znanja, naziv, čas šolanja,stopnja izobrazbe, usmeritev, poklic,...) je relacija, ki združuje več vrst entitet:šolsko izobrazbo, znanje (npr. predmet določenega izobraževalnega programa), tečaj, seminar ipd.

S.ZNANJE (šifra znanja,šifra znanja, vrsta povezave,...) podaja strukturo za relacijo ZNANJE: za šole, tečaje predmete iz njihovega programa, vertikalno povezovanje šol različnih stopenj (možnost nadaljnega šolanja) ipd.

TEČAJ (šifra znanja,planiran datum izvedbe,...) je relacija, ki omogoča načrtovanje in spremljanje izvedbe raznih, predvsem internih izobraževalnih oblik.

POKLIC (šifra poklica, naziv,...) pojasnjuje poklic pridobljen z določeno izobraževalno obliko in omogoča razlikovanje med obema pojmoma.

PTEST DNO (šifra p.testa,šifra DNO, mejne vrednosti,...) pove katere teste morajo opraviti zaposleni na določenem delavnemu mestu in v katerih mejnih vrednostih mora biti rezultat.

PTEST DELAVEC (šifra p.testa,delavska številka, rezultat,...) podaja rezultate posameznih merjenj s fiziološkimi testi.

ZNANJE DNO (šifra znanja,šifra DNO,...) je relacija, ki pove katera znanja so zahtevana za opravljanje posameznih DNO.

ZNANJE DELAVEC (šifra znanja,delavska številka, datum začetka šolanja, datum zaključka,dosežen uspeh,...) vsebuje podatke o družinskih članih zaposlenih delavcev.

SOCIALNA SKUPINA (delavska številka,šifra skupine, datum vpisa, kategorija,...) vsebuje informacijo, ki omogočajo nekatere analize s socialnega področja, kar je pomembno za določene zunanje ustanove pa tudi za ustrezno obravnavanje in varovanje pravic delavcev znotraj organizacije. Združitev teh podatkov v posebni relaciji omogoča tudi boljšo zaščito (tajnost) podatkov s tega področja.

KRITERIJ STAN.TOČKOVANJA (šifra, naziv,utež,...) predtsvlja seznam kriterijev, ki jih upoštevamo pri stanovanjskem točkovanju. Utež je podatek, ki pove v kolikšni meri se upošteva kriterij pri izračunu skupnega števila točk.

STAN.TOČKE (delavska številka,število točk,...)

MKB (šifra MKB,naziv, grupa, število obolelih, oznaka interaktivnosti, signalna vrednost,...) predstavlja šifrant bolezni po mednarodni klasifikaciji bolezni. V to relacijo so vključeni tudi podatki (naprimer trenutno število obolelih), ki jih sicer lahko izpeljemo iz relacije OBOLELI To je redundanca za katero pa smo se odločili, da omogočimo interaktiven dostop do teh podatkov. To omogoča delavcem v obratni ambulanti pravočasno reagiranje na razne pojave epidemij ali zdravju škodljivih sprememb v delavnem okolju.

OBOLELI (delavska številka,MKB,dat., dat zaključka bolovanja, vzrok,...) vsebuje informacijo o obolelih delavcih. Te informacije se po zaključku bolovanja in obdelavi s področja OD brišejo, razen za tiste šifre MKB, ki imajo to (v relaciji MKB) posebej označeno. S tem je mogoče kvalitetnejše obravnavanje bolnikov s kroničnimi boleznimi.

2. FUNKCIONALNI MODEL

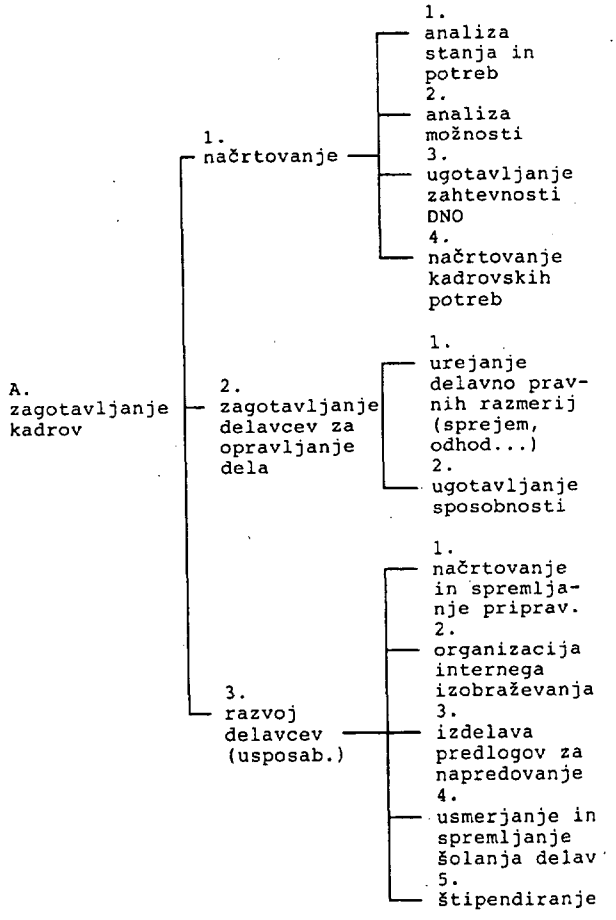
Poglavje o funkcionalnem modelu je razdeljeno v dva razdelka. Prvi prikazuje razgradnjo kadrovske dejavnosti. Prikaz je enostaven - s pomočjo drevesne strukture, povezave, ki obstajajo med posameznimi funkcijami niso prikazane. To pomanjkljivost delno odpravlja drugi razdelek, ki v matrični obliki podaja povezavo med podatkovnim in funkcionalnim modelom.

2.1 Drevesna shema in opis posameznih funkcij

Prvi nivo razgradnje kadrovske dejavnosti kot smo ga ugotovili v ŽJ predstavlja spodnja shema:

kadrovska dejavnost-
- A zagotavljanje kadrov
- B varovanje pravic delavcev
- C oblikovanje poročil za zunanje organizacije

A. Zagotavljanje kadrov zajema kadrovsko dejavnost v ožjem pomenu: načrtovanje potreb, razporejanje in razvoj oziroma usposabljanje delavcev. Podrobnejšo razdelitev kaže shema:



Slika 3: Razgradnja funkcije "zagotavljanje kadrov"

A.1.1 Analiza stanja in potreb sloni na podatkih o DNO in njihovi zasedenosti, njen rezultat pa je pregled DNO, ki so neustrezno zasedene bodisi glede na normativ (predvideno število delavcev) bodisi glede na zahtevane sposobnosti. V relaciji DNO so vpisane tudi delavne naloge in opravila, ki jih šele načrtujemo (glede na organizacijske spremembe ali nove investicije). V tem primeru analiza daje potrebe po novih kadrih oziroma pregled delavcev, ki bodo po ukinitvi določenih DNO ostali nerazporejeni.

A.1.2 Analiza možnosti ugotavlja sposobnosti delavcev (znanje,psihološke, fiziološke sposobnosti), kandidatov za sprejem in štipendistov, ugotovitve primerja z rezultati analize stanja in potreb ter oblikuje predloge razporeditve, premestitve in dodatnega usposabljanja delavcev. Ena od nalog, ki spadajo v ta sklop funkcij je izdelava "optimalnega" predloga razporeditve s pomočjo metod linearnega programiranja.

A.1.3 Ugotavljanje zahtevnosti DNO je, kot pove že ime,namenjeno ugotavljanju sposobnosti, ki jih mora imeti delavec razporejen na določeno delavno nalogo oziroma opravilo. Vloga informacijskega sistema pri zagotavljanju te funkcije je omejena predvsem na vnos in shranjevanje podatkov o sposobnostih (znanja, testi), zahtevnost posameznih DNO ter izpis teh podatkov po različnih kriterijih, iskanje sorodnih DNO in določanje zahtevnosti sestavljenih DNO.

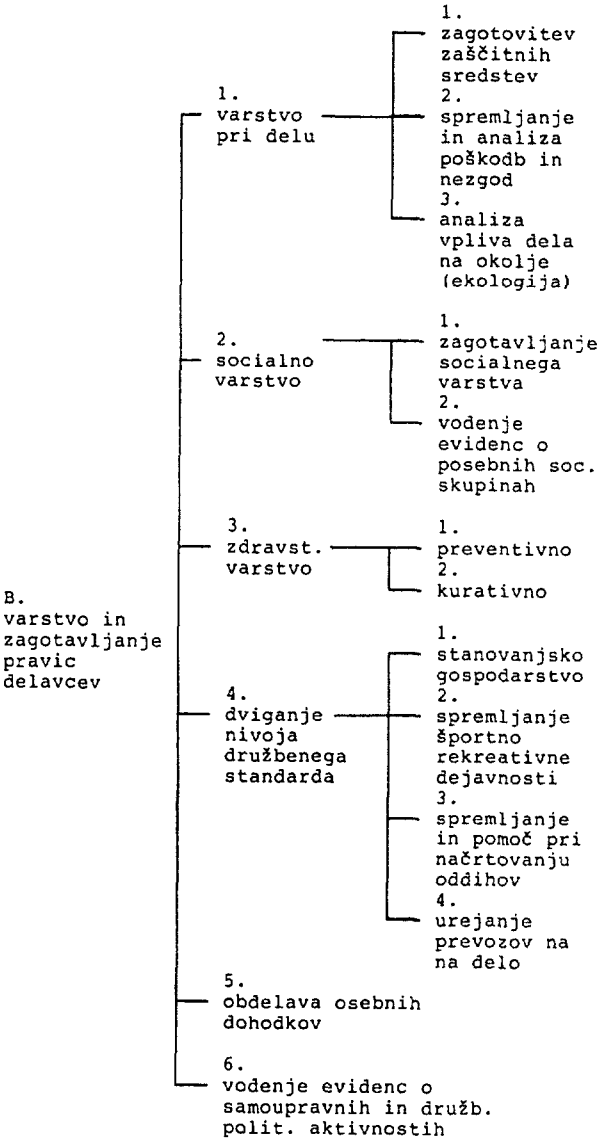
A.1.4 Načrtovanje kadrovskih potreb in analiza kadrovskih gibanj je funkcija namenjena oblikovanju predvsem takih informacij, ki pomagajo pri odločanju na strateškem nivoju: kakšna je fluktuacija, kateri so vzroki zanjo, kakšno bo predvidoma gibanje kadrov v posameznih obdobjih in kakšne bodo potrebe (izdelava projekcij), kako se spreminja izobrazbena struktura kadrov, kakšna je fluktuacija po posameznih profilih.

A.2 Zagotavljanje delavcev za opravljanje dela je sklop funkcij, ki podpirajo predvsem operativne naloge na ožjem področju kadrovske dejavnosti: vnos podatkov o novosprejetih delavcih, popravljanje podatkov o delavcih, izpis odločb o razporeditvi oziroma premestitvi, vnos podatkov o štipendistih in kandidatih za sprejem, oblikvoanje razpisov za prosta delavna mesta. Večina teh nalog se bo izvajala interaktivno, kar je za določena opravila na operativnem nivoju nujno.

A.3 Razvoj delavcev je funkcija, ki jo informacijski sistem podpira predvsem z: vodenjem evidenc o možnostih nadaljnega izpopolnjevanja in delavcih vključenih v razne oblike izobraževanja, analizo potreb po usposabljanju, s pomočjo pri spremljanjem izvajanja pripravništva ipd. Sem spada tudi spremljanje inovativne dejavnosti delavcev in oblikovanje predlogov o možnem napredovanju glede na uspehe pri usposabljanju.

B. Informacijsko podporo zagotavljanju in varstvu pravic delavcev smo razdelili v naslednje podfunkcije (glej sliko 4):

B.1 Prvi sklop funkcij s področja varstva pravic je namenjen večanju varnosti in zmanjševanju škodljivih vplivov pri opravljanju posameznih DNO. Tudi tu je vloga informacijskega sistema omejena predvsem na shranjevanju podatkov in oblikovanju poročil: vodenje evidenc o zaščitnih sredstvih potrebnih na posameznih DNO, o oskrbljenosti delavcev s temi sredstvi, o nezgodah in vzrokih zanje, o vrsti in teži poškodb pri nezgodah, o škodljivih vplivih v posameznih okoljih in o delavnih mestih z benificirano delavno dobo. Zahtevnejša naloga pa je analiza teh podatkov s ciljem odkrivanja sredin z večjo pogostostjo nezgod, njihovih



Slika 4: Razgradnja funkcije "varstvo in zagotavljanje pravic delavcev"

značilnosti ter preverjanje hipotez o vzrokih in pogojih, ki privedejo do nezgode.

B.2 je sklop nalog namenjen zagotavljanju socialnega varstva delavcev ter hranjenju in obdelavi podatkov o posebnih socialnih skupinah, informacijski sistem pa podpira operativne naloge oddelka, ki se ukvarja s tem področjem in oblikuje razne statistične analize.

B.3 Ena pomembnejših funkcij v sklopu kadrovske dejavnosti je zagotavljanje zdravstvenega varstva, tako kurativnega kot preventivnega. Predvidoma naj bi obdelave s tega področja omogočale poleg običajnega izračuna nadomestil za osebni dohodek tudi:

- različne analize o obolelosti delavcev
- pravočasno detekcijo naraščanja obolevnosti z določeno diagnozo. Tovrstne informacije so se izkazale kot pomembne za odpravljanje vzrokov bolovanj. (V DO Sava-Kranj so s spremljanjem informacij te vrste dosegli pomembne uspehe pri zmanjševanju obolevnosti (6))

- učinkovito spremljanje in obravnavo delavcev s kroničnimi obolenji
- izdelavo predlogov premestitev za delavce z zmanjšano delavno sposobnostjo in oblikovanje programa rehabilitacije oziroma prekvalifikacije
- izdelavo programov preventivnih pregledov z upoštevanjem zdravstvene zahtevnosti delavnih mest in ogroženosti posameznih delavcev
- analizo vpliva posameznih delavnih okolij na zdravje
- interaktivni vpogled v zdravstveno zgodovino posameznega delavca

B.4 Podfunkcije združene v funkciji "dviganje družbenega standarda" so dovolj pojasnjene s samo shemo. Posebej omenimo le naloge informacijskega sistema na področju stanovanjskega gospodarstva. POleg vodenja evidenc o bivalnih pogojih delavcev, spada v to področje tudi oblikovanje prednostnih list za dodelitev stanovanjskih pravic in izračun višine kreditov za stanovanjsko gradnjo. Glede na veliko število prosilcev smatramo, da je računalniška podpora upravičena.

B.5 Obdelava osebnih dohodkov je bolj ali manj taka, kot jih poznamo v drugih delavnih organizacijah. Vključenost v sistem za podporo kadovski dejavnosti, uporaba sodobnejše podatkovne baze in interaktivni vnos podatkov tudi temu področju prinaša večjo prožnost in ažurnost, možnost kvalitetnejših analiz ter manj podvajanja pri vnosu podatkov.

B.6 Samoupravno komuniciranje je področje, ki ga do sedaj nismo podpirali z računalnikom. Predlagani model predvideva vsaj najnujnejše: vodenje podatkov o samoupravnih organih in organih družbeno političnih organizacij, vodenju podatkov o vključitvi delavcev v te organe, vodenju podatkov o delu delegacij in enostavnejše obdelave teh podatkov.

C. V skupini "izdelava poročil za zunanje organizacije" smo združili, kot pove ime, oblikovanje zakonsko določenih ali drugače dogovorjenih poročil oziroma statističnih analiz za potrebe raznih zunanjih organizacij, kot so zavod za statistiko, skupščina občine, sis za zaposlovanje itd.

2.2 Povezava med funkcionalnim in podatkovnim modelom

Povezava med obema predlaganima modeloma je prikazana v matrični obliki (tabela 1). Vrstice v matriki predstavljajo funkcije iz funkcionalnega modela, stolpci pa relacije iz podatkovnega modela. Oznake na presečiščih vrstic in stolpcev v matriki pa povedo, kakšen odnos velja med funkcijo (v vrstici) in relacijo (v stolpcu). Pri tem je pomen posameznih oznak naslednji:

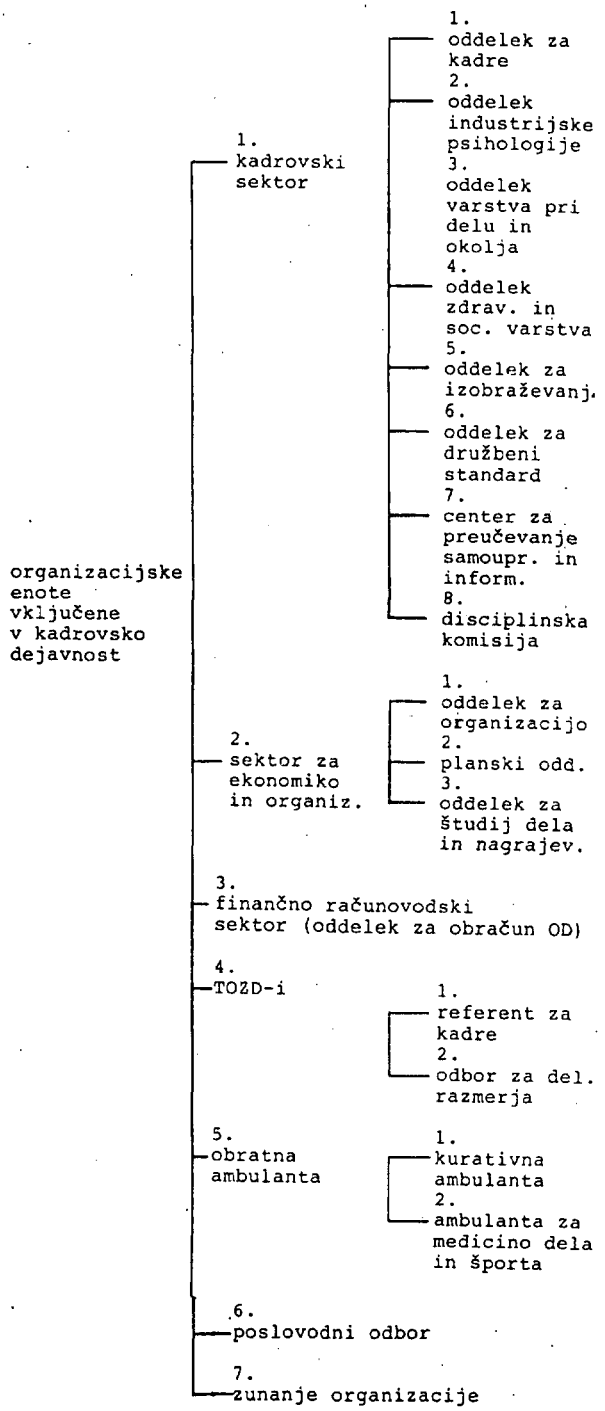
V ... spraševanje - funkcija le sprašuje po vrednostih podatkovnih elementov (atributov) v relaciji

K ... kreiranje -funkcija vpisuje nove n-terice (vrstice) v relacijo

M ... spreminjanje - funkcija spreminja vrednosti podatkovnih elementov (atributov) v relaciji.

3. VLOGA ORGANIZACIJSKIH ELEMENTOV

Realizacija obeh predlaganih modelov bo bolj ali manj vplivala na delo in naloge posameznih organizacijskih enot. V tretjem poglavju poskušamo vsaj grobo prikazati ta vpliv s pomočjo dveh matrik: prva prikazuje odgovornost organizacijskih enot do vpisovanja oziroma čitanja podatkov v bazi podatkov, druga pa odgovornost pri izvajanju funkcij.
Organizacijske enote, vključene v kadrovsko dejavnost so prikazane s spodnjo drevesno shemo (glej sliko 5).



Slika 5: Organizacisjki elementi vključeni v kadrovsko dejavnost

funkcije \ relacije	DELAVEC	DNO	ORG. ENOTE	ODLOČBA	PTEST	PTEST, DNO	PTEST, DELAVEC	FTEST	FTEST, DNO	FTEST, DELAVEC	ZNANJE	ZNANJE, DNO	ZNANJE, DELAVEC	POKLIC	TEČAJ	ZAŠČIT. SREDSTVA	FUNKCIJE	FUNKCIJE, DELAVEC	DRUŽINSKI ČLANI	POSEBNE SOC. SKUP.	KRITERIJ STAN.TČK.	KRITERIJ, DELAVEC	ŠIFRA MKB	ŠIFRA MKB, DELAVEC	OBOLELI
A.1.1 Analiza stanja in potreb		V	V	V	V	V	V	V	V	V	V	V	V												
2 Analiza možnosti		V	V	V	V	V	V	V	V	V	V	V	V												
3 Ugotavljanje zahtev. DNO		M			K	K		K	K		K	K													
4 Načrtovanje potreb		K	K		M	M		M	M		K	K													
A.2.1 Urejanje del. prav. razm.	K			K			M			M			M						K						
2 Ugotavljanje sposobnosti					K		K	K		K	K		K												
A.3.1 Pripravništvo	M	V	V	K																					
2 Interno izobraževanje	V	V	V	V							K	K	K	K	K										
3 Predlogi za napredovanje	M	V	V	K																					
4 Šolanje delavcev											V	V	K		M										
5 Štipendiranje	K										K		K												
B.1.1 Zaščitna sredstva	V	V														K									
2 Poškodbe, nezgode	V	V	V	V															K					K	K
3 Ekologija								K	K	V															
B.2.1 Socialno varstvo	M																	K					V	V	V
2 Posebne socialne skupine	V																		K						
B.3.1 Preventivno zdravstvo	M	V	V	V				V	V	K														K	K	K
2 Kurativno zdravstvo	M	V	V	V																				K	K	K
B.4.1 Stanovanjsko gospodarstvo	V																				K	K			
2 Rekreacija, šport	M																								
3 Oddihi	M																								
4 Prevozi	M																								
B.5 Osebni dohodki	M	V	V	M																					
B.6 Samoupravne aktivnosti	V	V	V	V													K	K							
C Poročila za zunanje org.	V	V	V	V				V	V	V	V	V	V				V	V	V	V			V	V	V

tabela 1: Povezava med podatkovnim in funkcionalnim modelom (matrika transakcij)

Poleg kadrovskega sektorja, ki je pri izvajanju kadrovkse dejavnosti vsekakor najpomembnejši, imajo pri izvajanju kadrovske dejavnosti nemajhno vlogo tudi drugi organizacijski elementi, česar pri načrtovanju ustreznega informacijskega sistema nismo smeli zanemariti. Predlagani sistem prinaša predvsem več neposrednosti: mnoge od informacij, ki jih posamezne enote (službe,oddelki) dobijo s pomočjo oddelkov kadrovskega sektorja, naj bi s pomočjo informacijskega sistema oblikovali samostojno. Podobno velja tudi za zbiranje podatkov: posamezni oddelki jih bodo vnašali s pomočjo računalnika neposredno v informacijski sistem.
Primer takega povečanja neposrednosti je spremljanje in obdelava bolniškega staleža. V obstoječem sistemu obratna ambulanta za vsakega obolelega izpiše obrazec z osnovnimi podatki, ta obrazec dopolnijo na oddelku za socialno in zdravstveno varstvo (kadrovski sektor),na oddelku za obračun OD (finančno računovodski sektor) pa na podlagi podatkov na njem oblikujejo novega, primernega za obdelavo na računalniku. Obratna ambulanta prejme od oddelkov kadrovskega sektorja različne analize obolevnosti. V novem sistemu bodo osnovne podatke vnašali interaktivno, jih (tudi interaktivno) dopolnjevali na ostalih oddelkih, informacijski sistem pa bo poleg obdelav s področja OD zagotovil tudi oblikovanje različnih informacij o obolevnosti.

3.1 Odgovornost pri oblikovanju in uporabi podatkov

je prikazana z matriko v tabeli 2. Vrstice predstavljajo posamezne elemente organizacisjke sheme, stolpci pa relacije podatkovnega modela. Oznake na križišču stolpcev in vrstic pomenijo:

V ... organizacijski element le sprašuje po vrednostih atributov relacije

K ... organizacijski element je odgovoren za vpis ali spreminjanje vrstic relacije

M ... organizacijski element spreminja vrednost atributov

relacije / organizacijske enote	DELAVEC	DNO	ORG. ENOTE	ODLOČBA	PTEST	PTEST, DNO	PTEST, DELAVEC	FTEST	FTEST, DNO	FTEST, DELAVEC	ZNANJE	ZNANJE, DNO	ZNANJE, DELAVEC	POKLIC	TEČAJ	ZAŠČIT. SREDSTVA	FUNKCIJE	FUNKCIJE, DELAVEC	DRUŽINSKI ČLANI	POSEBNE SOC. SKUPINE	KRITERIJ STAN. TČK	KRITERIJ, DELAVEC	ŠIFRA MKB	ŠIFRA MKB, DELAVEC	OBOLELI
1.1 Oddelek za kadre	K	M	V	K	V	V	V	V	V	V	V	V	K	V	V	V			K	V			V	V	V
1.2 Oddelek ind. psihologije	V	V	V	V	K	K	K	M	V	V	M	M	V	V	V					V			V	V	V
1.3 Oddelek varstva pri delu	V	M	M	V	M	M	V	M	M	V	M	M	V	V	M	K				V			V	V	V
1.4 Odd. zdr. in soc. varstva	M		V	V															M	K			M	V	M
1.6 Oddelek za družbeni stand.	M	V	V	V																V	K	K			
1.5 Oddelek za izobraževanje	M	V	V	V	V	M	V				K	K	K	K	K				V						
1.8 Disciplinska komisija	M	V	V	V																V					
1.7 CPSI, samoupr. org. DPO	V	V	V	V													K	K							
2.1 Oddelek za organizacijo		K	K	V		V		V	V		M	M		V											
2.2 Planski oddelek		M	V																						
2.3 Oddelek za študij de.,in ngr.		M	M	V	V	V		V	V		M	M		V											
3 Oddelek za obračun OD	M	V	M	M																V		V			M
4.1 Referent za kadre TOZD	M	V	V	M	V	V	V	V	V	V	M	M	V	V	V	M				V					
4.2 Odbor za del. razmerja	M	V	V	V							V	V	V	V						V					
5.1 Kurativna ambulanta	M	V	V	V			V	V	V	V										M			K	K	K
5.2 Ambulanta za med. dela	M	M	M	V	M	M	M	K	K	K			V	V		M				V			K	K	K
6 Poslovodni odbor		V	V								V	V													
7 Zunanje organizacije	V	V		V								V	V	V			V	V		V					

tabela 2: Matrika odgovornosti pri oblikovanju in uporabi podatkov

3.2 Odgovornost za izvajanje funkcij

je prikazana z matriko, podobno matriki odnosa do podatkov. Stolpci predstavljajo funkcije iz funkcionalnega modela. Posamezne oznake pomenijo:

O ... organizacijska enota je odgovorna za izvajanje funkcije

S ... organizacijska enota sodeluje pri izvajanju funkcije

Matrika odgovornosti za izvajanje funkcij je prikazana s tabelo 3.

4. ZAKLJUČEK

Predlagani model prinaša v primerjavi z obstoječim nekaj bistvenih sprememb:

1. Podatkovni model je v primerjavi z obstoječo bazo podatkov mnogo bogatejši. Poleg osnovnih podatkov o delavcih vsebuje tudi podatke o delavnih nalogah in opravilih, sposobnostih oziroma znanjih, podatke o zdravju delavcev, podatke s področja ekologije, socialnega varstva, družbenega standarda itd. Poleg tega ustrezna podatkovna baza ne bo vsebovala le podatkov, ki odražajo trenutno stanje, ampak tudi podatke o preteklih stanjih in za nekatere entitete tudi predvideno bodoče stanje (načrtovanje DNO na primer).

2. Datoteke obstoječe baze podatkov niso v tretji normalni obliki (posledica mnogih dopolnjevanj), kar povzroča znane težave pri vzdrževanju sistema. Tudi to nevšečnost odpravlja nov model, ki smo ga oblikovali upoštevajoč postopke normalizacije.

3. Posebno prožnost predlaganega modela predstavljajo tudi "strukturnice" za posamezne entitete. Te nosijo informacijo o že odkritih povezavah med posameznimi entitetami iste vrste poleg tega omogočajo enostavno vpeljavo novih povezav, različne interpretacije entitet istega tipa ali pa enotno obravnavo sicer sorodnih, ne pa povsem istovrstnih entitet.

4. Predvideni sistem bo interaktiven v kar največji možni meri. To iz dosedanjih opisov ni posebej razvidno, zato dodajamo: interaktivni so vsi postopki vnašanja podatkov (v okolju kjer le-ti nastajajo) in vpogledi, ki ne zahtevajo več kot 20-30 pristopov do različnih zapisov (vrstic) v bazi podatkov. Tak pristop zagotavlja, kot je to pač značilno za interaktivne sisteme, večjo ažurnost, neposrednejšo uporabo informacijskega sistema, manj podvajanj določenih postopkov (izpolnjevanja obrazcev in prepisovanja njihove vsebine na pomnilne medije), predvsem pa hiter dostop do informacij.

funkcije \ organizacijske enote	ODDELEK ZA KADRE	ODDELEK IND. PSIH.	VARSTVO PRI DELU	ZDR. IN SOC. VARSTVO	ODD. ZA DRUŽB. STAND.	IZOBRAŽEVANJE	DISCIPLINSKA KOMISIJA	C P S I	ODDELEK ZA ORGANIZACIJO	PLANSKI ODDELEK	ODD. ZA ŠTUDIJ DE. IN ČA.	ODDELEK ZA OBRAČUN OD	REFERENT ZA KADRE	ODBOR ZA DEL. RAZMERJA	KURATIVNA AMBULANTA	AMBULANTA ZA MED. DELA	POSLOVODNI ODBOR	ZUNANJE ORGANIZACIJE
A.1.1 Analiza stanja in potreb	O	S								S								
2 Analiza možnosti	O	S		S		S			S	S	S					S		
3 Ugotavljanje zahtev. DNO		S	S			S			S		O					S		
4 Načrtovanje potreb	O								S	S	S							
A.2.1 Urejanje del. razmerij	O			S			S					S		S				
2 Ugotavljanje sposobnosti	O	S				S							S		S	S		
A.3.1 Pripravništvo	O												S					
2 Interno izobraževanje		S				O							S					
3 Predlogi za napredovanje	O	S											S	S				
4 Šolanje delavcev		S	S			O							S					
5 Štipendiranje						O												
B.1.1 Zaščitna sredstva			O															
2 Poškodbe, nezgode				O											S	S		
3 Ekologija		S	O								S					S		
B.2.1 Socialno varstvo	S			O								S						
2 Posebne socialne skupine	S			O			S											
B.3.1 Preventivno zdravstvo	S														S	O		
2 Kurativno zdravstvo				S								S			O			
B.4.1 Stanovanjsko gospodarstvo				S	O													
2 Rekreacija, šport					O													
3 Oddihi					O													
4 Prevozi					O													
B.5 OD				S								O			S			
B.6 Samoupravne aktivnosti								O										
C Poročila za zunanje org.	O	S	S	S	S	S		S	S	S		S			S	S	S	S

tabela 3: Matrika odgovornosti za izvajanje funkcij

Model kakršen koli pač je, ne more dokazati svojih kvalitet, dokler se ne izkaže kot slika nekega konkretnega, uspešno delujočega sistema. Kljub temu, da opisani model take potrditve še nima, menim, da so opisane rešitve lahko v pomoč vsem, ki se ukvarjajo z načrtovanjem računalniško podprtih informacijskih sistemov za podporo kadrovski dejavnosti. Predvsem to velja za skrbno načrtovan podatkovni model, saj se nabor podatkov s področja kadrov od ene do druge (delavne) organizacije ne razlikuje kaj dosti.

Uporabljena literatura in gradivo:

(1) J. Martin: Principles of Data-Base Management, (Prentice - Hall, Inc, 1976)

(2) C. J. Date: A Guide to DB2 (Addison - Wesley, 1983)

(3) M. Krisper, F. Žerdin: Informacijsko inženirstvo (FE, Ljubljana, 1985)

(4) I. Svetlik in sodelavci: Naloge kadrovske dejavnosti v OZD (dopisna delavska univerza Univerzum, Ljubljana, 1978)

(5) B. Čeh, I. Ščavničar: Informacijski sistem preventivnega zdravstvenega varstva (interno gradivo ŽJ, 1985)

(6) J. Vadnjal: Računalniški model spremljanja in analize obolevnosti in bolniškega staleža v tovarni Sava Kranj (Sava Kranj, Kranj 1983)

(7) S. Alagić: Relacione baze podataka, (Svijetlost Sarajevo, 1984)

JEDNO PROŠIRENJE BIBLIOTEKE R A T F O R - a POTPROGRAMIMA ZA OBRADU ALFANUMERIČKIH VELIČINA

INFORMATICA 2/87

Toma Janoš
Institut za mernu tehniku i upravljanje,
Fakultet tehničkih nauka u Novom Sadu

UDK 681.3.068

PRIKAZ: U ovom radu je prikazan jedan pristup obradi alfanumeričkih veličina na programskom jeziku FORTRAN. Potprogrami za realizaciju ovog pristupa napisani su na programskom jeziku preprocesora za FORTRAN, RATFOR-u, i testirani su prilikom izrade kros-asemblera za mikroprocesor Z80, koji je takodje pisan na RATFOR-u.

ABSTRACT: The paper describes an approach in processing alphanumerical values using FORTRAN, and a realisation of this approach through subroutines written in RATFOR, the FORTRAN preprocessor. These subroutines are tested in a cross-assembler for Z80, also written in RATFOR.

U V O D

U ovom radu je izložena jedna od primena ideje da se efikasnost pisanja programa povećava primenom gotovih i proverenih opštih i funkcijskih potprograma /1/. Ovaj pristup implicira da se, pri pisanju potprograma iz nekog većeg skupa programa, pojedini potprogrami pišu tako da budu medjusobno nezavisni i što opštije namene.

Potprogrami pisani po ovom načelu mogu se vrlo efikasno koristiti pri razvoju drugih programa i predstavljaju programski alat, koji više korisnika može da primeni pri rešavanju problema odredjenog tipa.

Pošto je RATFOR (racionalni FORTRAN) preprocesor, koji je ugradjen na računaru Ei-H6/53 u Institutu za mernu tehniku i upravljanje, izveden tako da može programe iz sistemske biblioteke da uključuje putem INCLUDE naredbi /3/, u razvijani skup programa, proširenje biblioteke RATFOR-a potprogramima opšte namene značajno olakšava pisanje novih programa. Ovim se vreme potrebno za razvoj i testiranje softvare-a bitno skraćuje, što smanjuje troškove i povećava efikasnost primene digitalnog računara.

Potprogrami koji su prikazani u ovom radu su proistekli iz primene RATFOR-a (odnosno FORTRAN-a) na nenumeričke obrade, prilikom izrade kros-asemblera za mikroprocesor Z80, i odnose se prvenstveno na rad sa alfanumeričkim nizovima - string-ovima.

Osnovne postavke o upotrebi string-ova u RATFOR-u date su u prvom poglavlju, u kojem je opisan i skup ranije postojećih potprograma iz biblioteke RATFOR-a. Ovi potprogrami su poslužili kao osnova za proširenje opisano u drugom poglavlju, gde je svaki opšti ili funkcijski potprogram dodat u RATFOR biblioteku detaljno opisan. Algoritmi potprograma su dati u obliku pseudokoda.

1. PREDSTAVLJANJE ALFANUMERIČKIH VELIČINA

1.1. Uobičajeni pristup obradi alfanumeričkih veličina

Kod primene računara za nenumeričke obrade važan je način komuniciranja programa sa okolinom, to jest način unosa i ispisa podataka u/iz datoteka računarskog sistema. Kod uobičajene upotrebe FORTRAN-a ulaz i izlaz podataka, pa i alfanumeričkih veličina, vrši se u pakovanom obliku /2/. Ovaj pristup nije najpogodniji za pisanje programa kojima se obradjuje neki tekst, jer su redovi u tekstu obično različite dužine i sa različitim brojem reči.

Kod pisanja kros-asemblera ovakav način unosa i ispisa vrednosti u programskom jeziku FORTRAN (ne treba zaboraviti da je RATFOR preprocesor za FORTRAN) nije najpogodniji, pošto su obrade uglavnom na nivou jednog alfanumerika.

1.2. Predstavljanje alfanumeričkih veličina u obliku string-ova

Da bi se proširile mogućnosti primene FORTRAN-a i na nenumeričke obrade u biblioteku RATFOR-a je ugradjeno nekoliko potprograma kojima je omogućen unos i ispis jednog alfanumerika na datoteku (file) sa odredjenim logičkim brojem (lfn). Osnovu ovog pristupa čine dva potprograma /1/:

GETCH - koji iz datoteke, odredjene logičkim brojem, unosi jedan alfanumerik u ulazno/izlazni buffer i

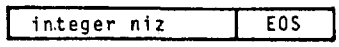
PUTCH - koji u ulazno/izlazni buffer datoteke odredjene logičkim brojem, upisuje jedan alfanumerik ili, ako je taj alfanumerik NEWLINE, ispisuje sadržaj buffer-a na datoteku.

Očigledno, ovakav pristup podrazumeva da se unos i ispis podataka obavlja isključivo preko alfanumeričkih veličina, što treba imati u vidu kod ulazno/izlaznih operacija.

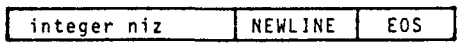
Veličina polja u ulazno/izlaznom redu ovde nije odredjena nekim formatom, pa je označavanje kraja string-a moralo biti rešeno drugačije. Izabrano je rešenje, preporučeno u /1/, da se kraj string-a označi nekom posebnom vrednošću EOS (end of string) koja ne pripada skupu znakova računara i na računaru Ei-H6/53 iznosi -2. Znakove veličine se u RATFOR-u pamte kao celobrojne veličine (INTEGER). Tako, string u

*) Radjeno u okviru postdiplomskih studija, na predmetu RAČUNARI.

RATFOR-u ima oblik



Da bi se niz alfanumerika, koji čini jedan red u datoteci, razlikovao od RATFOR string-a, na njegov kraj se pre znaka EOS stavlja znak NEWLINE iz skupa znakova računara. Dakle, niz alfanumerika, koji čine jedan red ima oblik



Ovako zamišljeno komuniciranje programa sa okolinom omogućava onome ko piše program veliku slobodu u oblikovanju unosa i ispisa podataka. Medjutim, ovo takodje podrazumeva i pretvaranje svih veličina, koje treba ispisati, u nepakovani oblik, odnosno RATFOR string, kao i pretvaranje unetih veličina u nepakovanom obliku u pakovani, ako je to potrebno. Za pretvaranje veličina iz jednog u drugi oblik potrebno je, dakle, obezbediti odredjen skup potprograma, koji bi svi korisnici RATFOR-a imali na raspolaganju.

Odredjen broj potprograma sa ovom namenom je postojao u biblioteci RATFOR-a. Ovi potprogrami, koji čine osnovu proširenja RATFOR biblioteke opisanog u ovom radu, biće objašnjeni u sledećem odeljku.

1.3. Osnovni skup potprograma iz RATFOR biblioteke za rad sa string-ovima

Potprogrami iz RATFOR biblioteke, koji će ovde biti opisani, se, prema nameni, mogu podeliti u tri grupe. Prvoj pripadaju potprogrami za unos i ispis podataka: getch, putch, getlin, putlin, outnum. Drugu grupu čine potprogrami za rukovanje znakovnim nizovima: equal, length, scopy, a treću potprogrami za pretvaranje veličina iz jednog oblika u drugi: itoc, cvtstr, strcvt.

Prikaz namene i upotrebe ovih potprograma dat je prema navedenoj podeli.

1.3.1. Potprogrami za unos i ispis podataka

1.3.1.1. Integer function getch (c, lfn)

Ovaj funkcijski potprogram kao svoju vrednost i vrednost argumenta c vraća sledeći alfanumerik iz datoteke sa logičkim brojem lfn. Ime getch mora biti navedeno u opisnim naredbama pozivnog programa kao INTEGER. Način pozivanja potprograma je

```
char = getch (c, lfn).
```

1.3.1.2. Subroutine putch (c, lfn)

Ovaj potprogram upisuje alfanumerik c u ulazno/izlazni buffer. Po upisivanju znaka NEWLINE slog upisan u buffer se upisuje u datoteku sa logičkim brojem lfn. Način pozivanja potprograma je

```
call putch (c, lfn).
```

1.3.1.3. Integer function getlin (line, lfn)

Ovaj funkcijski potprogram celobrojnog tipa upisuje red iz datoteke sa logičkim brojem lfn u string line. Kao svoju vrednost potprogram vraća broj elemenata string-a ili vrednost EOS, ako je unet prazan red. Ime getlin mora biti navedeno u opisnim naredbama pozivnog programa kao INTEGER. Način pozivanja potprograma je

```
len = getlin (line, lfn).
```

1.3.1.4. Subroutine putlin (line, lfn)

Ovaj opšti potprogram upisuje red, sadržan u stringu line, na datoteku sa logičkim brojem lfn. Način pozivanja potprograma je

```
call putlin (line, lfn).
```

1.3.1.5. Subroutine outnum (n, len, lfn)

Ovaj opšti potprogram pretvara broj u pakovanom obliku u RATFOR string sa prikazom u decimalnom brojnom sistemu i ispisuje ga u polje dužine len u ulazno/izlazni buffer datoteke sa logičkim brojem lfn. Ako je dužina prijemnog polja pozitivan broj, broj n se pri ispisu poravnava desno, ako je len negativno poravnava se levo, a ako je len=0 nema poravnavanja. Način pozivanja je

```
call outnum (n, len, lfn).
```

1.3.2. Potprogrami za rukovanje string-ovima

1.3.2.1. Integer function equal (str1, str2)

Ovaj funkcijski potprogram uporedjuje dva RATFOR string-a. U opisnim naredbama pozivnog programa mora biti naveden kao INTEGER. Ako su string-ovi jednaki, equal prima vrednost YES, inače je NO. Vrednosti YES i NO korisnik može odrediti u DEFINE naredbi RATFOR-a /3/. Način pozivanja potprograma je

```
ind = equal (str1, str2).
```

1.3.2.2. Integer function length (str)

Ovaj funkcijski potprogram vraća kao svoju vrednost dužinu RATFOR string-a str, bez znaka EOS. Način pozivanja potprograma je

```
len = length (str).
```

1.3.2.3. Subroutine scopy (start,from,target, to)

Ovaj opšti potprogram preslikava RATFOR string start od znaka sa indeksom from u string target od znaka sa indeksom to. Način pozivanja potprograma je

```
call scopy (start,from,target,to).
```

1.3.3. Potprogrami za pretvaranje stringova

1.3.3.1. Integer function itoc (int,str,size)

Ovaj funkcijski potprogram pretvara celobrojnu veličinu int u pakovanom prikazu u njen nepakovani prikaz (RATFOR string) str u decimalnom brojnom sistemu. Parametar size je najveća dimanzija string-a str. Kao svoju vrednost ovaj potprogram vraća dužinu string-a. Način pozivanja potprograma je

```
len = itoc (int,str,size).
```

1.3.3.2. Integer function cvtstr (holl,str)

Ovaj funkcijski potprogram pretvara tekst u znacima navoda holl u RATFOR string. Kao svoju vrednost potprogram vraća dužinu RATFOR string-a. Način pozivanja potprograma je

```
ind = cvtstr (holl,str).
```

1.3.3.3. Subroutine strcvt (str,delim,holl)

Ovaj opšti potprogram pretvara RATFOR string str u FORTRAN prikaz holl, tipa CHARACTER. Način upotrebe potprograma je

call strcvt (str,delim,holl).

Proširenje RATFOR biblioteke novim potprogramima, opisano u narednom poglavlju, ima za osnovu izložene osnovne postavke i opisane već postojeće potprograme.

2. PROŠIRENJE BIBLIOTEKE RATFOR-a

Potprogrami, kojima je proširen opisani osnovni skup potprograma RATFOR biblioteke, su napisani prilikom izrade kros-asemblera za mikroprocesor Z80. Ovo je nametnulo i neke posebne zahteve.

2.1. Nenumeričke obrade pri izradi kros-asemblera

Obrada teksta koja je potrebna kod izrade kros-asemblera ima odredjene specifičnosti. To su, pre svega, potreba za predstavljanjem brojeva u raznim brojnim sistemima (binarno, oktalno, decimalno i heksadecimalno), za izdvajanjem reči odredjene dužine iz reda izvornog asemblerskog programa, za umetanje jednog string-a pre, u sredinu i iza drugog string-a itd. Da bi se zahtevi za ovakvim obradama zadovoljili, bilo je potrebno razviti odgovarajuću programsku podršku. Detaljan opis potprograma koji ispunjavaju i ovakve zahteve dat je u daljem tekstu.

2.2. Skup potprograma kojima je proširena RATFOR biblioteka

Potprogrami za obradu alfanumeričkih veličina, koji su dodati u biblioteku RATFOR-a se, prema nameni, mogu podeliti slično ranije navedenoj podeli već postojećih potprograma. Dakle, dodati su potprogrami za unos i ispis podataka: nextw, outi, douti, dfouti; potprogrami za rukovanje RATFOR string-ovima: firstc, addstr, fill, prefil, midstr, skipbl, cmphr i potprogrami za pretvaranje iz jednog oblika u drugi: itos, stoi, ditos, dstoi. Navedeni potprogrami su detaljno opisani po ovim grupama.

2.2.1. Potprogrami za unos i ispis

2.2.1.1. Integer function nextw (in,i,limit, out)

Ovaj funkcijski potprogram uzima iz string-a in, koji je unet pomoću potprograma getlin, narednu reč ograničene dužine. Ovde je reč definisana kao niz alfanumerika ograničen znakovima TAB ili BLANK. Reč koja je izdvojena iz in se smešta u string out, a indeks string-a in, promenljiva i, se postavlja da pokazuje na prvi alfanumerik iza unete reči. Nextw dobija vrednost dužine reči, ako je ona kraća od vrednosti limit, inače dobija vrednost ERR, koju korisnik može definisati RATFOR naredbom DEFINE /3/. Tok algoritma je

```
integer function nextw (in,i,limit,out)
   call skipbl (in,i) ! preskakanje vodećih
                      ! TAB i BLANK
   while (element in(i) nije EOS ni BLANK)
      prenesi alfanumerik u out(i)
   return
end.
```

2.2.1.2. Subroutine outi (n,l,out,base)

Ovaj opšti potprogram ispisuje veličinu n u pakovanom obliku, u buffer datoteke sa logičkim brojem (lfn) out u obliku string-a sa prikazom u brojnom sistemu sa osnovom base. Veličina l odredjuje podešavanje broja u izlaznom polju. Ako je l manje od 0, broj se podešava u levo a polje se dopunjava BLANK-ovima; ako je l=0, nema podešavanja ni dopune BLANK-ovima; pri l veće od 0 se broj podešava u desno sa vodećim BLANK-ovima. Tok algoritma je

```
subroutine outi (n,l,out,base)
   ! pretvaranje u RATFOR string
   len=itos (n,str,MAXSTRING,base)
   if (treba popuniti vodećim BLANK-ovima)
      popuna BLANK-ovima
   ispis nepakovanog prikaza broja
   if (treba popuniti BLANK-ovima iza broja)
      popuna BLANK-ovima
   return
end.
```

2.2.1.3. Subroutine douti (n,l,out,base)

Ovaj potprogram je isti kao i outi, osim toga što je celobrojna veličina, koju treba ispisati, u dvostrukoj tačnosti. Ovo je potrebno ako je, na računaru Ei-H6/53, ceo broj veći, po apsolutnoj vrednosti, od 32767.

2.2.1.4. Subroutine dfouti (n,l,out,base)

Ovaj potprogram ispisuje ceo broj u dvostrukoj tačnosti u polje dužine l u buffer datoteke sa logičkim brojem out. Ispis je sa osnovom brojnog sistema base. Polje se popunjava sa leve strane vodećim nulama. Tok algoritma je

```
subroutine dfouti (n,l,out,base)
   ! pretvaranje broja u RATFOR string
   len = ditos(n,str,MAXSTRING,base)
   if (treba ispisati vodeće nule)
      ispis vodećih nula
   ispis nepakovanog prikaza broja
   return
end.
```

2.2.2. Potprogrami za rukovanje string-ovima

2.2.2.1. Integer function firstc (str,c)

Ovaj funkcijski potprogram nalazi indeks prvog elementa string-a str, koji je jednak c. Firstc vraća indeks pronadjenog elementa kao svoju vrednost ili, ako c nije nadjeno vraća vrednost ERR. Tok algoritma je jednostavan, pošto se indeks odredjuje pretraživanjem string-a str, pa neće biti prikazan.

2.2.2.2. Integer function addstr (body,str)

Ovaj funkcijski potprogram dodaje string str na kraj string-a body, tj. spaja dva string-a. Kao svoju vrednost addstr vraća ukupnu dužinu izlaznog string-a. Tok algoritma je

```
integer function addstr (body,str)
   ! pronalaženje indeksa elementa koji je
   ! jednak EOS
   for (body(i) ^=EOS)
      ;
   ! dodavanje str na body
   for (str(j) ^=EOS)
      dodaj str na body
   return
end.
```

2.2.2.3. Subroutine fill (i,hexcod,len,singlc)

Ovaj potprogram popunjava string hexcod, od znaka i+1 do ukupne dužine len, alfanumerickima singlc. Tok algoritma je

```
subroutine fill (i,hexcod,len,singlc)
   for (nije dostignuto len)
      dodavanje singlc na hexcod
   return
end.
```

2.2.2.4. Subroutine prefil (str,len,singlc)

Ovaj potprogram popunjava string str sa leve strane vodećim alfanumericima singlc, do ukupne dužine len. Tok algoritma je

```
subroutine prefil (str,len,singlc)
   if (string je kraći od len)
      popuni sa singlc
   return
end.
```

2.2.2.5. Subroutine midstr (mid,i,j,body)

Ovaj opšti potprogram je namenjen za upisivanje string-a mid u string body, od njegovog i-tog do j-tog elementa. Ako je j veće od ukupne dužine body, za j se uzima vrednost dužine body. Tok algoritma je

```
subroutine midstr (mid,i,j,body)
   if (j) dužine string-a body)
      j = dužina string-a body
   for (nije dostignuto j)
      preslikaj mid u body
   return
end.
```

2.2.2.6. Subroutine skipbl (line,i)

Ovaj potprogram je namenjen za nalaženje prvog elementa string-a line različitog od BLANK i TAB. Tok algoritma je

```
subroutine skipbl (line,i)
   while (TAB ili BLANK)
      uvećati brojač i
   return
end.
```

2.2.2.7. Integer function cmphr (txt,str,i,j)

Ovaj funkcijski potprogram je namenjen za uporedjivanje niza alfanumerika u znacima navoda txt sa RATFOR string-om str, od njegovog i-tog do j-tog elementa. Ako su txt i string jednaki, cmphr prima vrednost YES, inače je NO. Tok algoritma je

```
integer function cmphr (txt,str,i,j)
   pretvori txt u RATFOR string
   if (jednakost)
      cmphr = YES
   else
      cmphr = NO
   return
end.
```

2.2.3. Potprogrami za pretvaranje iz nepakovanog u pakovani prikaz i obrnuto

2.2.3.1. Integer function itos (int,str,isiz, base)

Ovaj funkcijski program pretvara veličinu int iz pakovanog prikaza u nepakovani prikaz broja str(isiz) sa osnovom brojnog sistema base. Kao svoju vrednost itos vraća dužinu string-a str, a ako je postojala greška pri pretvaranju vraća vrednost ERR. Tok algoritma je

```
integer function itos (int,str,isiz,base)
   if (base nije u dozvoljenim granicama)
      itos = ERR
      return
! izračunavanje string-a
   repeat
      int se deli osnovom brojnog sistema
      i popunjava se str
   until (int == 0 ili je dostignuto isiz)
   odredjivanje znaka broja
   promena redosleda elemenata u str
   return
end.
```

2.2.3.2. Integer function stoi (str,i,int,base)

Ovaj potprogram je namenjen za pretvaranje nepakovanog prikaza broja str, sa osnovom brojnog sistema base, u pakovani oblik int, počev od i-tog elementa str. Kao svoju vrednost stoi vraća NOERR ako je pretvaranje uspelo, inače prima vrednost ERR. Tok algoritma je

```
integer function stoi (str,i,int,base)
   if (base izvan opsega)
      stoi = ERR
      return
   call skipbl(str,i) ! preskakanje praznih mesta
   minus0 = predznak ! odredjivanje predznaka
   repeat
      izračunavanje int
   until (str nije EOS, BLANK ili NEWLINE)
   return
end.
```

2.2.3.3. Integer function ditos (int,str,isiz,base)

Ovaj potprogram ima funkciju ekvivalentnu sa itos, samo što je veličina int u dvostrukoj tačnosti. Ovo je potrebno, na Ei-H6/53, kod pretvaranja veličina većih, po apsolutnoj vrednosti, od 32767 u nepakovani oblik. Tok algoritma je opisan kod potprograma itos.

2.2.3.4.Integer function dstoi (str,i,int,base)

Ovaj potprogram ima funkciju ekvivalentnu sa stoi, samo što je veličina int u dvostrukoj tačnosti. Ovo je potrebno, na računaru Ei-H6/53 kod pretvaranja nepakovanog prikaza brojeva većih po apsolutnoj vrednosti od 32767 u pakovani. Tok algoritma je opisan kod potprograma stoi.

ZAKLJUČAK

Potprogrami opisani u ovom radu predstavljaju deo podrške, koja omogućava skraćivanje vremena razvoja programa uz smanjenje broja grešaka, primenom FORTRAN preprocesora RATFOR.

Za očekivati je da razvijeni programi, kao i ovaj rad, mogu da posluže korisnicima kao osnova za dalje proširenje RATFOR biblioteke.Programi opisani u ovom radu su sastavni deo RATFOR biblioteke na računaru Ei-H6/53 u Institutu za mernu tehniku i upravljanje Fakulteta tehničkih nauka u Novom Sadu i mogu INCLUDE naredbom RATFOR-a biti uključeni u razvijani skup programa.

Autor se zahvaljuje prof. dr Danilu Obradoviću sa Instituta za mernu tehniku i upravljanje Fakulteta tehničkih nauka u Novom Sadu na korisnim sugestijama datim u toku pisanja rada.

LITERATURA

/1/ Kernighan, B.W., Plauger, P.J., A PROGRAMOZAS MAGASISKOLAJA (Software tools, Hungarian translation), Muszaki Konyvkiado, 1982.
/2/ Parezanović, N., RAČUNSKE MAŠINE I PROGRAMIRANJE, programski jezik FORTRAN IV, PFV, Beograd, 1979.
/3/ Kemenci, Z., RATFOR priručnik za upotrebu, Beograd, 1984.

RAZISKAVE RAČUNALNIKOV IN INFORMACIJE V NASLEDNJEM DESETLETJU

INFORMATICA 2/87

UDK 681.3.001.5

Anton P. Železnikar
ISKRA DELTA, Ljubljana

Ta članek, ki je bolj esejističen in manj strokovno usmerjevalen, je bil napisan kot osnovni motiv za akademsko razpravo o raziskovalnih možnostih na področju računalnikov in informacije v prihajajočem desetletju. V njem poudarja avtor na svoj način posebno usmeritev, katere cilj je graditev in uporaba inteligentnega paralelnega stroja.

Research of Computers and Information for the Next Decade.

This paper was prepared for an academic conversation on the possibilities of research in the field of computers and information in the coming decade. The paper reflects the autor's particular emphasis in which direction to proceed in order to reach the goal of construction and application of intelligent, parallel information machines.

1. Uvod

Razumevanje računalnikov in njihove uporabe (1) se razvija čedalje bolj očitno v smeri informacijskih strojev in njihovih procesov. Na nekaterih novih izhodiščih se oblikujejo pojmi, zamisli in realizacija prihodnjih

paralelnih oziroma
večprocesorskih računalniških sistemov in
njihovih inteligentnih lastnosti.

Ta ciljna izhodišča so osnovni motivi domala vseh današnjih državnih in podjetniških projektov novogeneracijske računalniške metodologije in tehnologije. Projekti s takšno usmeritvijo se izvajajo na Japonskem, v ZDA, Veliki Britaniji, ZR Nemčiji, Franciji, SZ, Indiji, Kitajski itd., začenjajo pa svojo razvojno pot tudi pri nas (v okviru razvojnih projektov Iskre Delte).

Raziskovalno, tehnološko in razvojno osnovo oblikujejo

paralelne masivne in
paralelne nemasivne večprocesorske (superračunalniške) računalniške arhitekture,
novonastajajoča metodologija umetne oziroma
strojne in programske inteligence in
zlasti nova filozofija oblikovanja razumevanja, konceptov, arhitekture, metodologije in tehnologije računalniških sistemov (1, 3),

ki spoznava nezadostnost racionalistične raziskovalne in razvojne tradicije pri koncipiranju inteligentnih strojev in inteligentnih programov.

2. Filozofija informacije

Inteligenca strojev in programov je uresničljiva le na ustrezni temeljni informacijski podstati, ki jo oblikujeta

informacijska nastajalnost in
spremenljivost informacijskih stanj in
procesov (3).

Informatičnost, ki je lastnost informacijskega nastajanja, je podlaga oziroma tehnološki imperativ inteligentnosti, tj. sposobnosti inteligentnega delovanja. V razdobju do leta 1990 in po njem bo brzkone mogoče razviti novo filozofijo informacije (3) do te stopnje, da bo iz nje že mogoče postopno graditi fundamentalno informacijsko teorijo; šele ta teorija bo lahko podlaga inteligenci in vrsti drugih višjih informacijskih funkcij, ki se bodo raziskovale tako v okviru tehnoloških kot živih informacijskih mehanizmov.

Že v okviru današnjih raziskovalnih in razvojnih razmišljanj je inteligenca le poseben informacijski pojav (informacijska kategorija), ki na določen način povezuje druge višje in nižje informacijske funkcije, kot so npr. motivacija, asociacija, strategija, ontološki model sveta, organizacija in različni značilni korteksni in tehnološki informacijski procesi. Inteligenca ostaja tako prej ko slej smiselni kompleks osnovnejših, toda tudi višjih informacijskih procesov. Tu se inteligenca razumeva kot visoko strukturirana in povezana informacija, kot zapleteno in prepleteno informacijsko procesno stanje oziroma procesnost.

Filozofija in teorija informacije in iz njiju izpeljivo razumevanje informacijskih strojev in programov bosta značilno interdisciplinarni raziskovalni dejavnosti; obe se bosta sklicevali, združevali in govorili v okviru več znanstvenih disciplin, kot so

nevralne znanosti (2) (nevrofiziologija, nevrofarmakologija, nevropsihologija, nevrolingvistika, nevroanatomija, genetika, biologija, biokemija, medicina itd.),
razumevanje obnašanja in učenja (eksperimentalna psihologija, psihiatrija, nevrologija),
informatika (razumevanje, spoznavanje in razumevanje spoznavanja živih in tehnoloških strojev in programov) in
informacijska tehnologija (nove arhitekture, metodologije, elementi).

3. Informacijski stroj

Tehnološki stroj ni v nobenem primeru posnetek živega stroja ali organizma. Avtomobil ni konj, letalo ni ptica, računalnik ne posnema možganov in robot ni bitje. Kljub temu postaja oblikovanje strojev vedno bolj povezano s filozofijo in znanjem o živem, npr. s filozofijo biti, izkustva in informacije in znanjem o možganskem ustroju, o nevrofizioloških procesih in o miselnih funkcijah. Večini strokovno vkalupljenih znanstvenikov in tehnologov se zdi povsmem neverjetno, da je lahko filozofsko razmišljanje praktično pomembno pri njihovem delu (1, 3). Trdne znanosti in tehnike, ki izhajajo iz evropske racionalistične tradicije, postajajo nemočne in nezadostne pri reševanju sodobnih informacijskih problemov. Ta pomanjkljivost se najizraziteje pojavlja v miselnosti in metodologiji umetne inteligence in pri raziskavah in razvoju novogeneracijskih strojev, za katere so projektni načrtovalci obljubljali in napovedovali inteligentno obnašanje.

Znanje in filozofija mišljenja o naravi biološkega bivanja, o jeziku in o naravi človekovega obnašanja postajata bistvena pri oblikovanju, gradnji in proizvodnji strojev in njihovih programov. Cilj današnjih oblikovalcev, načrtovalcev, graditeljev in tehnologov so

inteligentni stroji in
inteligentni programi.

Pri tem postaja jasno, da inteligence kot tehnološke lastnosti ne bo mogoče doseči brez temeljnih raziskav njene informacijske narave.

Informacijski stroj postaja tako tudi sinonim za inteligentni stroj. Na poti do inteligence stroja morata metodologija in tehnologija zagotoviti

informacijsko nastajalnost v tehnološki substanci (materialni arhitekturi novega računalnika) in
v pripadajoči informaciji (spremenljivem, metodološko nestrukturiranem programu).

Nastajalnost v okviru računalniške arhitekture je v bistvu dinamično, signalno, sporočilno ali vobče informacijsko krmiljena osnovna arhitektura: to je t. i.

dinamična strojna arhitektura.

Učinek takega krmiljenja je navidezna, virtualna arhitekturna spremenljivost z dejanskimi in potrebnimi dinamičnimi posledicami. Ta arhitekturna spremenljivost, ki je elementarna, je prav zaradi te elementarnosti modularna, podsistemska in naposled tudi sistemska, je tedaj spremenljiva v podrobnostih (v drobni tehnološki strukturi), elementarnih modulih (v srednji logični strukturi) in podsistemih (v grobi strojnojezikovni strukturi).

Novi računalniški sistemi bodo morali v okviru svojih paralelnih ali večprocesorskih arhitektur izpolnjevati tudi zahteve po določeni standarizaciji. Ti standardi se oblikujejo po dosedanjih izkušnjah kot posledice

tržne uspešnosti (administrativni oziroma poslovni računalniški sistemi) in
procesnih tehnoloških zahtev (procesni oziroma industrijski računalniški sistemi).

4. Informacijsko programiranje

Informacijski stroj z dinamično arhitekturo bo omogočil izvajanje t. i. informacijskih programov. Informacijski program je značilno

programirno nestrukturiran,
med svojim izvajanjem spremenljiv in
nastajajoč program.

Teh lastnosti današnji programi nimajo ali pa jih imajo le v značilno trivialni obliki. Informacijski program, ki bi bil inteligenten, je cilj današnje programirne metodologije. Ta metodologija je vklenjena v programirnorazvojna strukturna pravila, ki programsko funkcijo omejujejo in onemogočajo programsko nastajalnost že na ravni današnjih programirnih orodij, kot so prevajalniki in programski generatorji za visoke programirne jezike.

Razvoj masivnih paralelnih, nemasivnih večprocesorskih, novogeneracijskih računalnikih sistemov se sooča tudi s t. i.

programirno krizo.

Ko je nov oziroma novogeneracijski stroj arhitekturno realiziran, se pojavi najprej problem njegovega operacijskega sistema (sistema osnovnih sistemskih programov, ki omogočajo uporabo stroja) in njegove funkcionalne združljivosti s stroji prejšnje generacije. Razvoj in izdelava teh programov po kriterijih operativnosti in združljivosti se lahko pokažeta kot praktično nezadovoljiva ali celo nemogoča. Nekateri veliki razvojni projekti tako ne dosegajo ciljev, lastnosti, zmogljivosti in napovedi, ki so bile postavljene v projektnih načrtih. Cilji, ki se vobče ne morejo dosegati, so npr.

inteligentno obnašanje sistemov,
govorna komunikacija človek-stroj,
ekspertne lastnosti računalniških sistemov (primerljive z živimi izvedenci),
komunikacija v pisnem naravnem jeziku itd.

Programirna kriza je hkrati kriza umetne inteligence oziroma njene racionalistične metodologije (matematizacije, algoritmizacije, formalne jezikovnosti, znanstvene trdnosti).

Bistven element programirne krize je programska kompleksnost operacijskih in aplikativnih programov. Ta element kriznosti se praktično rešuje z razvojem programirnih orodij in seveda s projektnim vodenjem velikih programirnih skupin.

Pri novih projektih se pojavlja nekaj, kar je mogoče imenovati projektna kriza. Novi projekti na področju informacijskega stroja, informacijskega programiranja pa tudi informacijske tehnologije potrebujejo ob svojih začetkih posebno obravnavo z vidika

filozofije tehnološko in metodološko mogočega,

ne pa utemeljevanje projektne upravičenosti z napovedmi, ki jih kasneje ni mogoče uresničiti.

5. Informacijska tehnologija

Današnja informacijska tehnologija temelji na elektronskih integriranih vezjih, optičnih podatkovnih prenosnikih, elektromagnetnih, optičnih in elektromehaničnih pomnilnih in perifernih napravah, telekomunikacijski opremi itd. V integriranih vezjih prevladujejo še vedno polprevodniški in prevodniški elementi pri normalnih in nizkih temperaturah (supraprevodnost). Prihodnji razvoj se bo brzkone gibal tudi v smeri uporabe elektromagnetnih, antenskih, valovodnih pa tudi drugih, npr. bioloških elementov (npr. živčnih vezij), ki bodo integrirani v funkcionalna osnovna vezja.

Tehnološka osnova informacijskega stroja bo v marsičem inovativna. Uporaba elementov in procesov fizike trdne snovi, biokemije, genetike, bioloških in drugih substanc bo omogočala potrebno funkcionalno raznovrstnost v fizični, logični in uporabnostnojezikovni strukturi stroja. Z novo substančno in procesno strukturo se bo tako sestavljeni stroj približeval paralelni procesirni raznovrstnosti, ki je funkcionalno primerljiva z živo nevronsko substanco.

6. Živi in umetni sistemi

Osrednji živčni sistem ostaja tako od molekul življenja do korteksov zgleden primer smiselnega sistema, ki bi ga želeli po možnosti posnemati tudi s tehnološkimi stroji. Tu se že kaže nova interdisciplinarnost med nevralno in informacijsko znanostjo in nekaterimi obrobnimi področji (filozofija, psihologija, tehnologija). Umetna inteligenca je trenutno že na poti rekonstrukcije svojih osnov oziroma spoznavanje novih temeljev (1). Raziskave živih in umetnih sistemov postajajo vedno bolj medsebojno odvisne in pospešujoče. Oblikovanje interdisciplinarnih raziskovalnih projektov in skupin na akademskem in industrijskem področju je nujno, ker brez tega ni mogoče računati z raziskovalnimi rezultati, ki bi zagotavljali napredek na področju inteligentnih strojev in programov.

7. Sklep

Raziskave in razvoj računalnikov in informacije na novih, informacijskih in s tem na intelegenčnih temeljih se bodo intenzivneje nadaljevale v 90. letih. Današnji paralelni, superračunalniški in vektorski sistemi, ki v povezavi z današnjo umetno inteligenco in s svojimi funkcionalnimi značilnostmi napovedujejo t. i. peto računalniško generacijo, še ne bodo inteligentni. Vendar se bodo z njimi in z njihovo uporabo uveljavili nekateri standardi, kot so paralelne arhitekture, sistemska vodila, operacijski sistemi, komunikacijski in grafični standardi in nova področja njihove uporabe. S tem razvojem se bodo pojavili novi podjetniški in akademski projekti.

Računalniška industrija bo še nadalje glavni nosilec tega razvoja. Akademske raziskave, ki so pretežno začasne, obrobne in predvsem začetne (inicialne), bodo take ostale tudi v prihodnosti. Računalniška industrija mora tako prej ko slej računati z večjim delom investicij v razvoj

- novih sistemskih konceptov,
- načrtovalnih in programirnih metodologij,
- paralelne arhitekture,
- paralelnih operacijskih sistemov,
- sistemske inteligence,
- komunikacij,
- novih aplikacijskih področij itd.

Domača računalniška industrija realno v tem novem razvojnem kontekstu ne more pričakovati bistvenih ugodnosti na domačem trišču in bistvenega družbenega usmerjanja. Predvsem razvoj mednarodne razvojne in izvozne tržne strategije je lahko realni faktor njenega napredovanja.

8. Slovstvo

(1) T. Winograd, F. Flores: Understanding Computers and Cognition: A New Foundation for Design. Ablex Publ Corp, Norwood, NJ (1986).

(2) E. R. Kandel, J. H. Schwartz: Principles of Neural Science (Second Edition). Elsevier, New York (1985).

(3) A. P. Železnikar: Na poti k informaciji. Informatica 11 (1987), št. 1, 4-18.

JEZIK ZA SPECIFIKACIJU INFOLOŠKOG MODELA

INFORMATICA 2/87

Danilo Vidojković
Viša škola unutrašnjih poslova, Zemun
Vladan Jovanović
Fakultet organizacionih nauka, Beograd

UDK 681.3.069:519.688

U radu je predložen jezik za opis infološkog modela dobijenog Langeforsovim pristupom projektovanju informacionih sistema. Infološki model opisan predloženim jezikom moguće je preslikati u Yao-ov model za analizu troškova pristupa u fizičkoj organizaciji podataka.

AN INFOLOGICAL DATA MODEL SPECIFICATION LANGUAGE. In the article the language used for a specification of an infological data model acquired through Langefors's approach to information systems design is suggested. The infological data model described with the help of the suggested language is possible to map into Yao's attribute based model for database cost analysis.

1. U V O D

Projektovanje informacionih sistema predstavlja veoma kompleksan posao koji sadrži čitav niz različitih aktivnosti. Postoje mnoge metode za projektovanje informacionih sistema, ali ni jedna od njih ne pristupa sa jednakom sistematičnošću svim fazama projektovanja. Tako, uglavnom razlikujemo metode čije je težište na ranijim fazama projektovanja, vezanim za analizu korisničkih zahteva i izradu logičkog modela podataka, i metode koje više pažnje posvećuju projektovanju fizičke organizacije podataka na računaru. U ovom radu prikazan je jezik koji omogućava povezivanje dva takva pristupa: Langeforsovog pristupa projektovanju informacionih sistema[1] koji veliku pažnju posvećuje prvim fazama projektovanja, tako da se dobija logički model podataka (infološki model) koji verno odražava korisničke zahteve, i Yao-vog modela za analizu troškova pristupa u fizičkoj organizaciji podataka[6] koji uvodi jedinstven postupak proračuna vremena pristupa podacima u različitim organizacijama podataka i time omogućava optimalan izbor fizičke organizacije podataka. Infološki model opisan predloženim jezikom predstavlja polaznu osnovu za primenu Yao-vog modela, što omogućava zadovoljenje korisničkih potreba definisanih u infološkom modelu, uz optimalno iskorišćenje raspoloživih resursa.

U prvom delu rada prikazan je Langeforsov pristup projektovanju informacionih sistema, a u drugom delu dat je prikaz jezika za opis infološkog modela.

2. LANGEFORSOV PRISTUP PROJEKTOVANJU INFORMACIONIH SISTEMA

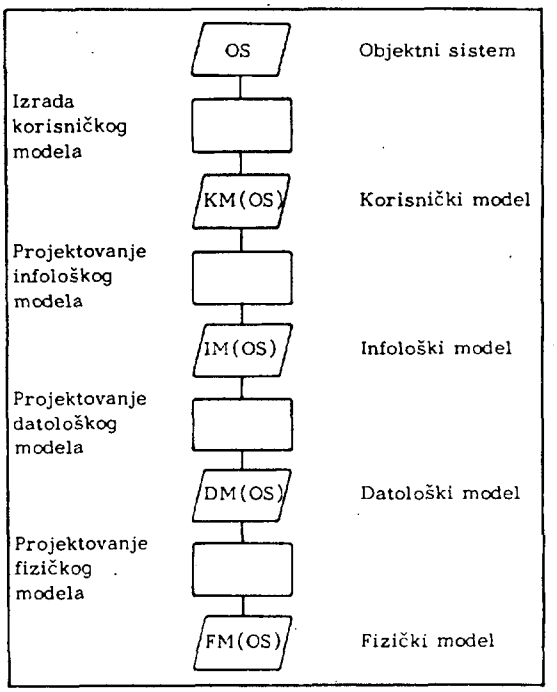
Osnovni cilj informacionog sistema je da ispuni informacione zahteve korisnika u skladu sa uslovima koji vladaju u objektnom sistemu. Medjutim, mnogi informacioni sistemi ne postižu taj cilj, pre svega zbog usko specijalističkog pristupa projektovanju informacio-

nih sistema. Često se u prvi plan stavljaju tehničke mogućnosti za izgradnju informacionog sistema što gotovo redovno dovodi do realizacije informacionog sistema koji ne zadovoljava u potpunosti potrebe korisnika. Tada je neophodno da se stvarne potrebe korisnika modificiraju da bi se uklopile u mogućnosti tako projektovanog informacionog sistema. Medjutim, kada se jave prevelike razlike izmedju mogućnosti informacionog sistema i potreba korisnika neophodno je izvršiti reprojektovanje što je, naravno, skopčano sa novim troškovima. Jasno i precizno definisani korisnički zahtevi u najranijim fazama projektovanja neophodan su preduslova za efikasan rad informacionog sistema.

Langeforsov pristup projektovanju informacionih sistema, poznat i kao infološki pristup, kao polaznu tačku upravo uzima potrebe krajnih korisnika planiranog informacionog sistema. Krajni korisnik je lice koje treba da reši problem odredjene vrste, ili da donese odluku u procesu upravljanja. Prema infološkoj teoriji informacioni sistem služi korisniku kao izvor informacija koje će mu pomoći da reši aktuelni problem. Sa tog aspekta sasvim je jasno da je od velikog značaja uključivanje korisnika u proces projektovanja informacionog sistema, bez obzira u kojoj meri je upoznat sa načinom rada računara i ostalih uredjaja koji će kasnije biti korišćeni.

Informacioni sistem koji se projektuje može se posmatrati kao model jednog dela realnog sveta (objektnog sistema). Projektovanje informacionog sistema predstavlja proces preslikavanja jednog dela objektnog sistema u informacioni sistem. Infološki pristup predlaže da se to preslikavanje izvrši sistematski, u četiri koraka:[4]

1. Preslikavanje objektnog sistema u korisnički model, OS → KM(OS);
2. Preslikavanje korisničkog modela u infološki model, KM(OS) → IM(OS);
3. Preslikavanje infološkog modela u datološki model, IM(OS) → DM(OS);
4. Preslikavanje datološkog modela u fizički model, DM(OS) → FM(OS).



Slika 1. Faze projektovanja informacionog sistema

Prva dava koraka čine infološku fazu projektovanja informacionog sistema. U ovoj fazi projektovanja potrebno je definisati tzv. spoljne karakteristike planiranog informacionog sistema. "Informacioni sistem, prema infološkom pristupu, treba da bude izgradjen tako da je dovoljno da korisnik specificira informaciju koju želi, na način koji njemu odgovara, a sistem će pronaći ili proizvesti podatke koji predstavljaju traženu informaciju, bez obzira na to kako su podaci memorisani i struktuirani u bazi podataka.[2] Za krajnjeg korisnika dovoljno je da sistem posmatra kao "rezervoar" u koji se smeštaju i iz koga se dobijaju odredjene vrste informacija. Spoljne karakteristike informacionog sistema odredjene su upravo mogućnostima prihvatanja i prezentiranja odredjenih vrsta informacija. Zadatak korisnika budućeg informacionog sistema je da u infološkoj fazi projektovanja utvrdi spoljne karakteristike informacionog sistema i na taj način jasno iskaže svoje informacione potrebe.

Tokom infološke faze projektovanja ne razmatraju se karakteristike resursa koji će se koristiti u fazi implementacije. To može, u odredjenim slučajevima da dove-

de do infološkog modela koga nije moguće realizovati raspoloživim sredstvima, Medjutim, i to je manji promašaj od onoga kada se, zbog preteranog prilagodjavanja karakteristikama uredjaja u ranim fazama projektovanja izgradi informacioni sistem koji nije u stanju da zadovolji potrebe korisnika. Infološki pristup predlaže da se tokom infološke faze samo u grubim crtama sagledaju karakteristike raspoloživih resursa i da se u tom svetlu projektuje infološki model.

Na osnovu precizno definisanog infološkog modela projektant je u mogućnosti da sprovede sledeću fazu projektovanja, datološku fazu. Tokom ove faze projektuju se datološke, odnosno unutrašnje karakteristike informacionog sistema. One obuhvataju datoteke i programe koji će realizovati informacione skupove i informacione procese definisane u infološkom modelu. Kao rezultat dobija se datološki model informacionog sistema. Tokom faze projektovanja datološkog modela uticaj performansi računara je znatno veći. U Langeforsovom pristupu naročita pažnja se posvećuje minimiziranju količine transporta podataka izmedju glavne i periferne memorije. U tom cilju se u datološkoj fazi vrši grupisanje (konsolidacija) datoteka i grupisanje procesa, što bitno utiče na konačnu strukturu datološkog modela. Treba, medjutim, naglasiti da ova prilagodjavanja ne utiču na karakteristike utvrdjene korisničkim i infološkim modelom.

Tokom realizacije prva tri koraka u procesu projektovanja informacionih sistema Langeforsovim pristupom definišu se koncepti prikazani u Tabeli 1, čime se utvrdjuje struktura infološkog modela.[2]

Elementarni koncept (e-koncept)
Elementarna poruka (e-poruka)
Deo za identifikaciju objekta (Object term)
Deo za identifikaciju svojstva (Property term)
Deo za identifikaciju vremena (Time term)
Elementarni proces
A/B - upit
Elementarni zapis
Elementarna datoteka
Glavna datoteka
Katalog
Katalog-datoteka kompleks

Tabela 1. Koncepti infološkog modela

Poslednji korak u procesu projektovanja informacionog sistema Langeforsovim pristupom treba da obuhvati postavljanje datološkog modela na računar, odnosno preslikavanje u fizički model. Ovaj korak nije u istoj meri precizan i sistematičan kao prethodni. Sistematizacija fizičkog projektovanja može se obezbediti samo preciznim zapisom željenih parametara (karakteristika) u formalnom jeziku. U tom cilju konstruisan je JEZIK ZA OPIS INFOLOŠKOG MODELA koji opisuje parametre infološkog modela relevantne za primenu Yao-vog modela za optimalan izbor fizičke organizacije podataka.

3. JEZIK ZA OPIS INFOLOŠKOG MODELA

Reči koje čine predloženi jezik proistekle su iz terminologije koja se koristi u infološkom modelu. Tako se katalog - datoteka kompleks u infološkom modelu označava sa DIRECTORY-FILE-COMPLEX, datoteka u kojoj su smešteni zapisi uredjeni po primarnom ključu sa MAIN-FILE, a katalozi koji služe za pristup podacima u datoteci, sa DIRECTORY-FILE. Veza izmedju pojedinih delova infološkog modela koji se nalaze na različitim hijerarhijskim nivoima opisuje se terminom WITH. Na primer, rečenica

DIRECTORY-FILE-COMPLEX vozilo WITH
MAIN-FILE dat-voz

govori da je datoteka "dat-voz" glavna datoteka katalog -datoteka kompleksa "vozilo".

Veza izmedju elemenata na istom hijerarhijskom nivou opisuje se terminom AND:

DIRECTORY-FILE-COMPLEX vozilo WITH
(MAIN-FILE dat-voz WITH
----------)
AND(DIRECTORY-FILE k-boja WITH
----------)

AND(DIRECTORY-FILE k-tip WITH
----------) .

Iz prikazanog opisa može se uočiti da katalog datoteka kompleks "vozilo" ima glavnu datoteku "dat-voz" i veći broj kataloga za pretraživanje datoteke.

Za označavanje zapisa u glavnoj datoteci i katalozima koristi se termin ENTRY. Termini za opis elemenata iz kojih se sastoji zapis analogni su usvojenoj terminologiji u infološkom pristupu: OBJECT-TERM opisuje objekat na koji se odnosi zapis u datoteci, PROPERTY-TERM definiše odgovarajuće svojstvo objekta, a TIME-TERM definiše vreme u kome naznačeni objekat ima odredjeno svojstvo. Opis jedne datoteke infološkog modela može tako da ima sledeći oblik:

```
MAIN-FILE dat-voz WITH
    ENTRY zap-voz WITH
            OBJECT-TERM reg-br
        AND PROPERTY-TERMS
                        marka
                    AND boja
                    AND tip
        AND TIME-TERM dat-reg .
```

Prilikom opisivanja zapisa kataloga mora se voditi računa da je katalog organizovan kao atributna datoteka, odnosno da je objekat o kome zapis informiše jedno od svojstava iz zapisa glavne datoteke. Za sledeće faze projektovanja značajno je da se u opisu infološkog modela bar približno odredi koliko objekat u zapisu kataloga ima mogućih vrednosti. To se prikazuje na sledeći način:

```
DIRECTORY-FILE k-boja WITH
    ENTRY z-boja WITH
        OBJECT-TERM boja WITH
            VALUE-SET v-boja(100) .
```

Iz prikazanog opisa vidi se da objekat "boja" (u stvari atribut) ima 100 različitih vrednosti. Za svaku konkretnu vrednost objekta o kome informiše zapis kataloga, kao svojstvo vezuje se niz objekata iz glavne datoteke. Na primer, ako belu boju imaju vozila BG238-345, BG32-33 i BG31-28, tada će zapis kataloga imati oblik:

```
/boja,bela/,
/reg-br,BG238-345/,
/reg-br,BG32-33/,
/reg-br,BG31-28/,
/dat-reg,120382/
```

I ovde je važno odrediti prosečan broj objekata iz glavne datoteke koji su u vezi sa objektima kataloga. Opis kataloga tako dobija sledeći oblik:

```
DIRECTORY-FILE k-boja WITH
    ENTRY z-boja WITH
            OBJECT-TERM boja WITH
                VALUE-SET v-boja(100)
        AND PROPERTY-TERM SEQ reg-br(1000) .
```

Izraz SEQ reg-br(1000) pokazuje da svakoj vrednosti OBJECT-TERM-a "boja" odgovara niz (SEQ) PROPERTY TERM-ova prosečne dužine 1000.

Osim opisa strukture modela podataka važno je opisati i transakcije koje će se nad bazom podataka realizovati. U infološkom pristupu transakcije su prikazane u formi $\mathcal{AB}$- upita tako da je i u jeziku za opis infološkog modela opotrebljena slična terminologija. Svi upiti iste vrste grupisani su u istu klasu upita. Opis svake klase upita sastoji se iz dva dela. U prvom su definisane promenljive, a u drugom delu su opisane aktivnosti koje su neophodne da bi se realizovao upit. Te aktivnosti su prikazane u obliku Bool-ovog izraza, gde definisane promenljive uzimaju odredjene konkretne vrednosti. Opis jedne moguće klase upita ima sledeći oblik:

```
QUERY-CLASS upit-1 WITH
        A-PROPERTIES
                boja
            AND marka
    AND     B-PROPERTIES
                ime-vl
            AND adr-vl

    AND ACTIONS
        FOR ALL zap-voz FROM dat-voz
        SUCH THAT (boja=bela OR boja=žuta)
                AND marka=Z101
        GET ime-vl AND adr-vl.
```

Kompletan opis infološkog modela ima oblik dat u Tabeli 2.

```
DIRECTORY-FILE-COMPLEX ime-kompleksa WITH

     (MAIN-FILE ime-gl-dat WITH
      ENTRY ime-zap WITH
            OBJECT-TERM ime-obj
       AND PROPERTY-TERM ime-sv-1
                     AND ime-sv-2

                     AND ime-sv-r
       AND TIME-TERM ime-vrem)

AND (DIRECTORY-FILE ime-kat-1 WITH
      ENTRY ime-k-zap WITH
            OBJECT-TERM ime-obj-1 WITH
              VALUE-SET skup-1(br-vred-1)
       AND PROPERTY-TERM SEQ niz-1(br-obj-1)
       AND TIME-TERM ime-vrem-1)


AND (DIRECTORY-FILE ime-kat-k WITH
     ENTRY ime-k-zap WITH
            OBJECT-TERM ime-obj-k WITH
              VALUE-SET skup-k(br-vred-k)
      AND  PROPERTY-TERM SEQ niz-k(br-obj-k)
      AND TIME-TERM ime-vrem-k)

AND (QUERY-CLASS ime-upita-1 WITH
           α-PROPERTIES ime-α-sv-1
                    AND ime-α-sv-2

                    AND ime-α-sv-m
       AND β-PROPERTIES ime-β-sv-1
                    AND ime-β-sv-2

                    AND ime-β-sv-j
       AND ACTIONS
             FOR ALL ime-zap FROM ime-gl-dat.
             SUCH THAT Bool-izraz(ime-α-sv-i)
             GET ime-β-sv-1 AND ime-β-sv-2...)


AND (QUERY-CLASS ime-upita-f WITH
             -------------
             -------------        )
END.
```

Tabela 2. Opis infološkog modela

4. ZAKLJUČAK

Jezik koji se predlaže u ovom radu omogućava projektantu da opiše sve parametre infološkog modela, dobijenog infološkim pristupom projektovanju informacionih sistema, koji su relevantni za izbor optimalne fizičke organizacije podataka pomoću Yao-vog modela.

Pomoću pogodnog algoritma moguće je preslikati opis infološkog modela u opis Yao-vog modela koji dalje služi kao polazna osnova za izbor optimalne fizičke organizacije podataka. Na taj način preslikani su zahtevi korisnika definisani u infološkom modelu u Yao-ov model. Izborom parametara karakterističnih za odredjenu fizičku organizaciju podataka projektant može dalje vršiti analizu troškova pristupa podacima i odabrati najpogodniju organizaciju i sa aspekta korisničkih zahteva i sa aspekta resursa koji se koriste za realizaciju informacionog sistema, što je predmet obradjen u /15/.

Predloženi jezik pruža mogućnost automatizacije postupka opisivanja infološkog modela i preslikavanja tog opisa u opis Yao-vog modela. Sobzirom na već utvrdjene mogućnosti automatizacije pojedinih faza projektovanja infološkog modela, kao i na mogućnost automatizacije analize troškova pristupa pomoću Yao-vog modela, predloženi jezik treba da bude još jedan korak ka potpunoj automatizaciji projektovanja informacionih sistema.

5. LITERATURA

1. B. Langefors: "Theoretical Analysis Of Information Systems", Studentlitteratur, Lund, 1970.

2. B. Langefors, B. Sundgren: "Information Systems Architecture", Petroclli/Charter, New York, 1975.

3. M. Lundeberg: "Four Important Problems in Analysis and Design of Information Systems", Systemeering 75, Studenlitteratur, Lund, 1975.

4. B. Sundgren: "Using The Infological Approach in The Design of Data Base", Systemeering 1975, Studentlitteratur, Lund 1975.

5. M.Lundeberg, G. Goldkuhl, A. Nilsson: "Information Systems Developement - A Systematic Approach", Prentice-Hall, New Jersey, 1981.

6. S.B. Yao: "An Attribute Based Model for Database Access Cost Analysis", ACM Transaction on Database Systems, Vol. 2, No. 1, March 1977.

7. T.J. Teorey and J.P. Fry: "Design of Database Structures", Prentice-Hall, 1982.

8. F.H. Lochovsky: "Data Models", Prentice-hall, New Jersey, 1982.

9. A. Albano, L. Cardelli, R. Orsini: "Galileo: A Strongly-Typed, Interactive Conceptual Language", ACM Transactions on Database Systems, Vol. 10, No. 2, June 1985.

10. J.D. Ullman: "Implementation of Logical Query Languages for Databases", ACM Transactions on Database Systems, Vol. 10, No. 3, Sept. 1985.

11. A. Borgida: "Language Features for Flexible Handling of Exeeptions in Information Systems" Acm Transactions on Database Systems, Vol. 10, No. 4, December 1985.

12. A. Borgida: "Features of Languages for Developement of Information Systems at The Conceptual Level", IEEE Software, January 1985.

13. G.H. Sockut: "A Framework for Logical-Level Changes Within Database Systems", Computer, Vol. 18, No. 5, May 1985.

14. L.A. Kaliničenko: "Metodi sredstva integracii neodnorodnih baz dannih", Nauka, Moskva 1983.

15. D. Vidojković: "Jedan pristup translaciji infološkog u Yao-ov model", magistarski rad u pripremi, Fakultet organizacionih nauka, Beograd.

UGOTOVITVE OB PREIZKUSU DEMONSTRACIJSKE VERZIJE PROGRAMSKEGA PAKETA MICRO-OPTRANS

Tadej Barle
Eli Delidžakova-Drenik
Ljubljanska banka – Združena banka
Ljubljana

UDK 519.713:519.688:681.3.06

POVZETEK - V Ljubljanski banki-Združeni banki smo imeli priložnost spoznati programski paket MICRO-OPTRANS, ki združuje funkcije preglednice, dela s podatkovnimi tabelami ter vsebuje orodje za razvoj ekspertnih sistemov. Preizkusili smo njegovo delovanje. V članku podajamo opis delovanja in naše ugotovitve. Smatramo da predstavlja zelo zmogljivo orodje za razvoj sistemov za pomoč pri odločanju.

ABSTRACT - In Ljubljanska banka-Združena banka we had the opportunity to test MICRO-OPTRANS, which is an integrated software package that includes a spreadsheet, a database, as well as an expert system development tool ("shell"). In the paper we describe the package and its functions. We think that is a very powerful tool in development of decision support systems.

UVOD

MICRO-OPTRANS je integriran programski proizvod, ki vsebuje preglednico, upravljanje s podatkovnimi tabelami, grafiko ter orodje za razvoj ekspertnih sistemov ("shell"). Omogoča razvoj sitemov za pomoč pri odločanju, zlasti na finančnem področju.

Deluje na mikroračunalnikih tipa IBM PC, XT in AT ali 100% kompatibilnih, pod operacijskim sistemom MS/DOS in zahteva najmanj 256kB spomina ter vsaj en disketni pogon.

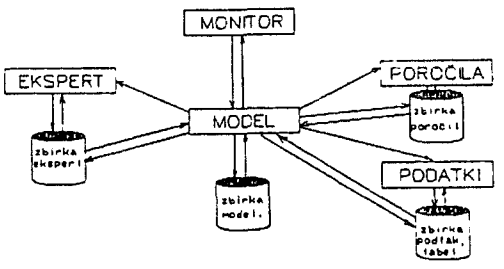
Programski paket MICRO-OPTRANS je izdelala francoska institucija SIG (Systemes informatiques de gestion) iz Jouy-en-Josas, ki se od ustanovitve leta 1970 ukvarja z razvojem in trženjem programskih orodij za pomoč pri odločanju. Največ njihovih uporabnikov je iz finančnega poslovnega področja. SIG sodeluje tudi z Institutom Jožef Stefan iz Ljubljane.

Za preizkus MICRO-OPTRANS-a smo imeli na voljo demonstracijsko disketo in priročnik z navodili za uporabo. Kot vsaka demonstracijska verzija programja , je imela tudi naša določene omejitve (izvajanje izključno na disketnem pogonu ter omejitve obsega podatkov in programov). Te so nas ovirale, da bi temeljiteje preverili delovanje in ocenili hitrost izvajanja. Vendar, kljub temu, smo se lahko seznanili s funkcionalnim delovanjem MICRO-OPTRANS-a. Proizvajalec tudi trži demonstracijsko verzijo v izobraževalne namene.

OPIS DELOVANJA

MICRO-OPTRANS sestoji iz nekaj segmentov oz. funkcionalnih celot z vodilnim segmentom na vrhu hierarhične strukture. Iz vodilnega segmenta - monitorja lahko preide uporabnik v naslednje vsebinsko opredeljene segmente:

- podatkovne tabele,
- modeli,
- poročila,
- ekspertni sistemi.



Slika 1. Struktura MICRO-OPTRANS-a

Segment podatkovne tabele omogoča delo s podatki. Tabele vsebujejo izključno numerične podatke, urejene po stolpcih in vrsticah. Velikost tabele je omejena na 32.766 vrstic in 512 stolpcev. Obseg podatkov je v mejah od -1 E+30 do 1 E+30 in je lahko podan z natančnostjo 11 signifikantnih znakov. Posamezen podatek je opredeljen je tudi s statusom : zanesljiv, ocenjena vrednost, ni na razpolago, ni vnešen.

Dostop do podatkovnih tabel v MICRO-OPTRANS-u je možen le preko modelov. Več modelov lahko uporablja isto podatkovno tabelo. S pomočjo modela podatkovna tabela dobi koordinate, podatki v tabeli pa simbolična imena in tudi smisel.

Model je osrednji del aplikativnega delovanja MICRO-OPTRANS-a. Uporabnik kreira modele s posebnim programskim jezikom, podobnim BASIC-u. V modelu uporabnik opredeli postopek dela s podatki in aktivira izbrano podatkovno tabelo. Z modelom aktivira prej definirana poročila ali pa ekspertne sisteme in jih povezuje s podatkovnimi tabelami.

Programski jezik pozna standarne algebarske in logične operatorje ter nekaj funkcij: SUM,AVG,MIN,MAX,VAR,STD ter LOG,EXP,ABS,SQRT. Interaktivno delovanje modela programiramo lahko z ukazi QUESTION in ANSWER. Komunikacijo z uporabnikom modela, za večjo preglednost, realziramo lahko z delitvijo zaslona po oknih. Krmiljunje znotraj programa izvajamo z IF...THEN in GOTO ukazi.

Segment "poročila" omogoča raznoliko oblikovanje in predstavitev podatkov tako na tiskalnik kot tudi na zaslon. Vnaprej pripravljena poročila uporabimo lahko z različnimi podatkovnimi tabelami. Možna je tudi grafična predstavitev podatkov v obliki krivulj, stolpcev ali nadstropnih stolpcev.

Najbolj zanimiv in zmogljiv del MICRO-OPTRANS-a je vsekakor segment ekspert. Segment ekspert je orodje za razvoj ekspertnih sistemov, tako imenovani "shell".

Predstavitev ekspertnega znanja je v obliki produkcijskih pravil:

ČE (pogoj) POTEM (sklep,akcija)

V pogojih se lahko sklicujemo na :

- podatke vsebovane v podatkovnih tabelah,
- vrednosti, ki jih je model izračunal,
- dejstva, katera so vsebovana v bazi znanja in podana z začetno vrednostjo, ter,
- dejstva, katerim uporabnik sistema v interaktivnem dialogu opredeli vrednost.

Sklepni del produkcijskega pravila prav tako dopušča več možnosti:

- izvedemo sklepanje (opredelimo dejstvo),
- zahtevamo aktiviranje drugege produkcijskega pravila z izbrano prioriteto,
- zahtevamo izvajanje modela (povratek v segment model, kjer izvedemo lahko dodatne izračune podatkov iz podatkovne tabele),
- zaključimo delovanje ekspertnega sistema.

Dejstva so dejansko parametri odločitvenega procesa. V bazi znanja jih podajamo kot spremenljivke, ki lahko zavzamejo eno od vrednosti iz vnaprej določenega nabora vrednosti ali pa vrednost "neznano". Tudi cilj odločitvenega procesa podamo kot dejstvo. Njegova inicialna vrednost je "neznano".

Preiskovanje baze znanja poganja mehanizem sklepanja (inference machine). Sklepanje poteka v smeri naprej, v smeri nazaj in tudi mešano. Delovanje mehanizma sklepanja je določeno s strukturo baze znanja. Mehanizem sklepanja normalno deluje v smeri naprej, dokler ne pride do produkcijskega pravila v pogoju katerega se sklicujemo na dejstvo z vrednostjo "neznano". Mehanizem začne delovati v smeri nazaj in poskuša opredeliti vrednost temu dejstvu. Če ne uspe, se ustavi in uporabniku postavi zahtevo da sam poda vrednost dejstvu. Potem mehanizem ponovno prične sklepanje v smeri naprej. Proces sklepanja se ustavi, ko je določena vrednost ciljnemu dejstvu.

Bazo znanja uporabnik kreira v preprostem programskem jeziku, upoštevati mora nezahtevno sintakso. Pred izvajanjem je potrebno bazo znanja prevesti (kompilacija).Velikost baze znanja je omejena z razpoložljivim spominom (200 produkcijskih pravil zavzame 64Kb spomina).

MICRO-OPTRANS je celovit sistem. Pod nadzorom monitorja opravljamo lahko pomožne funkcije (kreiranje in brisanje datotek, tiskanje, pregledovanje).Vsebina MICRO-OPTRANS-ovih datotek pa je nedostopna MS-DOS-u. Vendar pa vse vrste datotek (podatkovne tabele, modeli oz. programi in tudi ekspert) je možno konvertirati v tekstovno obliko. Prav tako MICRO-OPTRANS sprejme tekstovne datoteke, ki smo jih oblikovali z urejevalnikom besedila in jih pretvori v sebi primerno obliko.

UGOTOVITVE

MICRO-OPTRANS je integriran programski paket, ki vsebuje preglednico, upravljanje podatkovnih tabel, grafiko, urejevanik besedil ter orodje za razvoj ekspertnih sistemov. Ekspertni sistem tesno sodeluje s preglednico in podatkovnimi tabelami.

Preglednica MICRO-OPTRANS-a je zelo zmogljiva, 32.766 vrstic in 512 stolpcev, kar pomeni da omejitev postavlja razpoložljivi računalniški spomin. Medtem ko sta pri popularnih preglednicah tipa LOTUS 1-2-3 proceduralni in podatkovni del preglednice tesno povezana, tukaj sta ločena. Prav zato je spoznavanje MICRO-OPTRANS-a precej bolj zahtevno kot pa LOTUS-a 1-2-3. Lahko rečemo, da MICRO-OPTRANS ni orodje s katerim bi končni uporabnik, brez pomoči računalniškega strokovnjaka, kreiral preglednice.

Ločitev proceduralnega dela od podatkovnega ima vrsto prednosti:

- neodvisnost procedure od podatkov, oz.uporaba istega programa nad različnimi podatkovnimi tabelami,
- možnost strukturiranja: klic subrutine znotraj aktivnega modela, klic drugih modelov iz aktivnega modela,
- dokumentiranje preglednice,
- možnost dograjevanja in dopolnjevanja,
- neobčutljivost preglednice na neželjene nestrokovne posege.

Urejevanlik besedil je celozaslonski, podoben Wordstar-u. Primeren je za delo programerja, ne zajema pa funkcij avtomatizacije pisarniškega poslovanja.

Ekspertni del MICRO-OPTRANS-a je bil predmet največjega zanimanja v teku spoznavanja in preizkusa programskega paketa. Glede na to, da je bil to prvi produkt te vrste, ki smo ga spoznali, ga ne moremo primerjati z drugimi.

Vendar pa nezahtevna sintaksa za tvorbo baze znanja in demonstracijski primer, ki smo ga imeli na voljo, sta nas spodbudila, da smo izdelali testni primer in tako preizkušali delovanje sistema. Ugotovili smo:

-Struktura baze znanja t.j. zaporedje produkcijskih pravil in njihovo grupiranje v kriterije, vpliva na način delovanja mehanizma sklepanja. Ne vpliva pa na končni sklep. Zato je možno enostavno dopolnjevanje in dograjevanje baze znanja.

- Sklepni mehanizem deluje tudi ko je baza znanja nepopolna. če produkcijska pravila ne pokrivajo ves prostor možnih pojavov in sistem ne more določiti vrednost ciljnemu dejstvu, potem sistem postavi zahtevo uporabniku naj jo sam določi.

- Razlaga sklepanja, ki je pomembna komponenta ekspertnega sistema je tudi prisotna. V bazi znanja je možno vsakemu pravilu dodati sporočilo in komentar, ki ga sistem prikaže na zaslonu, ko je pravilo aktivirano. Prav tako, po končanem delu ekspertni sistem da lahko odgovor na vprašanje "zakaj?" oziroma razloži potek sklepanja.

- Preden izvajajamo ekspertni sistem, ga je potrebno prevesti (kompilacija). To nam omogoča varovanje baze znanja pred neželenimi posegi in tudi hitrejše izvajanje.

ZAKLJUČEK

MICRO-OPTRANS je zasnovan kot programsko orodje za razvoj sistemov za pomoč pri odločanju. V sistemih za odločanje, razvitih s pomočjo MICRO-OPTRANS-a sodelujejo deterministični modeli in algoritmi, ki uporabljajo egzaktne, vnaprej znane podatke ter ekspertni sistemi, v katerih je shranjeno znanje strokovnjakov.

Razvoj uporabnih sistemov za odločanje ni prav lahko delo. Zahteva sodelovanje vsaj dveh vrst ozko usmerjenih strokovnjakov : računalniških in stokovnjakov področja odločanja. še posebej naporno in težavno je uvajanje že izdelanih programskih rešitev na področju odločanja, ki je domena nenadomestljivih strokovnjakov.

Orodje kot MICRO-OPTRANS bistveno pospeši razvoj sistemov za odločanje. Prototipni način razvoja, ki je z njim mogoč, pa z večjo gotovostjo pripelje do uporabnih rešitev.

LITERATURA

1. Michel Klein: "Recent Developments in MICRO-OPTRNS : a KB/DSS Generator", EURO VIII, DSS and Expert Session, September 1986, Lisbon

2."MICRO-OPTRANS", Manual,Version 1.3, S.I.G., June 1986

3."MICRO-OPTRANS EXPERT", Manual, S.I.G.,June 1986

PRODUKT ZA UPRAVLJANJE RELACIJSKE BAZE PODATKOV – VAX Rdb/VMS

INFORMATICA 2/87

Marija Ipavec
Zavod SR Slovenije za družbeno planiranje
Center za informatiko

UDK 681.3.05

POVZETEK: Članek govori o produktu za upravljanje relacijske baze podatkov (VAX Rdb/VMS). V poglavju ena je opisana organizacija podatkov. Poglavje dva opisuje, kako se logična podatkova baza prevede v fizično z uporabo DEFINE stavkov in kako se podatkovno bazo rekonstruira. Poglavje tri govori o zaščiti podatkov v podatkovni bazi (uporabimo DEFINE PROTECTION stavek). To poglavje vsebuje še opis seznama za kontrolo dostopa do podatkovne baze. Četrto poglavje opisuje naslednje produkte za upravljanje baze podatkov: VAX CDD, VAX TDMS, VAX ACMS in VAX DATATRIEVE.

ABSTRACTS: The article is about relational database management product VAX Rdb/VMS that uses the relational model of data base organization. Chapter one describes data organization. Chapter two describes how to translate the logical database into a physical database using DEFINE statements and tells how to restructure an existing database. Chapter three shows how to define protection for your database using the DEFINE PROTECTION statement. This chapter includes a discussion of access control lists (ACLs). Chapter four describes information management products which includes: VAX CDD, VAX TDMS, VAX ACMS, VAX DATATRIEVE.

0. UVOD

Ustrezna organizacija podatkov v informacijskih sistemih je ključnega pomena za njihovo uspešno delovanje. Za tekoče vodenje, obdelovanje in izkazovanje različnih podatkov so se razvili sistemi za upravljanje z bazami podatkov-DBMS (Data Base Management System). Ti sistemi danes omogočajo hiter in enostaven dostop do podatkov. Ta programska oprema vključuje sodobne pripomočke za vnos, obdelavo in izkazovanje podatkov. Digitalov programski paket VAX Rdb/VMS je namenjen za upravljanje relacijske baze podatkov. Ta model baze podatkov nam omogoča, da lahko postavljamo vprašanja o objektih, o odnosih med objekti, kateri so atributi objekta itd..

Programski paket VAX/RDB VMS vsebuje naslednje funkcije:

- osnovni RDB paket za kreiranje in logično črpanje relacijskih zbirk podatkov,
- CDD podatkovni slovar za poenotenje, lažjo kontrolo in lažje koriščenje vseh vrst podatkov,
- TDMS orodje za lažje in učinkovitejše delo z ekranom,
- DTR produkt za kreiranje in črpanje poljubnih zbirk podatkov z vključeno osnovno grafiko,
- ACMS za lažjo in hiterejšo izdelavo raznih enkratnih in stalnih aplikacij.

Odlike tega paketa so naslednje:

- hitrost dostopa do baze podatkov (Rdb/VMS skrbi,da se Rdb/VMS stavki izvedejo v optimalnem času.),
- razumljiva podatkovna struktura (Podatki so shranjeni v tabelah.),
- interaktiven test (Interaktivni RDO pomožni program nam omogoča, da najdemo napako, če smo ukaz napačno zapisali.),
- popolna zaščita podatkov (Če pride do strojne ali programske napake, Rdb/VMS skrbi za BACKUP, za zapis dogodkov in za restavriranje baze podatkov.),
- vse definicije so shranjene centralno (v podatkovnem slovarju).

Podatke shranjene v Rdb/VMS podatkovni bazi lahko obdelujemo tudi s programi, ki jih napišemo sami v višje programskih jezikih, (BASIC, COBOL, FORTRAN, PASCAL). V te programe lahko vključimo Rdb/VMS stavke za ravnanje s podatki. Te stavke moramo v programu posebej označiti z RDB. Za dostop do baze podatkov uporabljamo stavek INVOKE, za shranjevanje podatkov uporabljamo stabek STORE, za spreminjanje podatkov uporabljamo stavek

MODIFY, za brisanje uporabljamo ERASE itd..
S stavkom

RDB INVOKE DATABASE FILENAME PERSONNEL

smo dobili dostop do baze podatkov z imenom PERSONNEL. Stavek INVOKE postavimo kjerkoli v programu, toda preden se začnemo sklicevati na druge Rdb/VMS stavke za ravnanje s podatki (velja za programske jezike BASIC, FORTRAN, PASCAL). V programu, ki je napisan v programskem jeziku COBOL, se stavek INVOKE nahaja v WORKINGSTORAGE SECTION (znotraj DATA DIVISION).
Prednost možnosti uporabe programov napisanih v višje programskih jezikih je predvsem v:
- hitrosti izvajanja (Podatke lahko dopolnjujemo in spreminjamo v nočnih urah, ko računalnik ni obremenjen z drugimi posli.),
- izboljšanem vhodu in izhodu podatkov na zaslon terminala (Za interaktivni vnos podatkov itd. uporabljamo program z VAX TDMS aplikacijo.).

1. ORGANIZACIJA PODATKOV

VAX Rdb/VMS je niz programskih orodij za upravljanje relacijskega modela baze podatkov. Pod modelom razumemo predstavitev določenih objektov in odnosov med temi objekti. Model nastane v postopku abstrakcije realnega sveta. Iz realnega sveta izberemo objekte. Vsak objekt ima lahko več atributov, s katerimi predstavimo lastnosti objektov.
Primer objekta delavec:

delavec (matična številka, ime, priimek, OD)

Objekt delavec ima štiri atribute: matično številko delavca, ime delavca, priimek delavca in osebni dohodek delavca. Dva različna delavca tvorita dve različni četvorki. Matične številke morajo biti tiste, ki enolično določajo delavce. Ta objekt predstavimo z n-torkam v tabelah. Kolone tabel odgovarjajo atributom, vrstice pa n-torkam vrednosti atributov.

Objekt delavec prestavljen v tabeli:

Matična številka	ime	priimek	osebni dohodek
941	Peter	Peterle	175040.
385	Marko	Kos	155045.
2430	Ines	Vadnal	98505.
764	Draga	Drole	305045.

Množica objektov je lahko povezana med seboj.

Primer povezane množice objektov:

OZD (šifra OZD, ime OZD, kraj OZD)

delavec (matična št. delavca, ime in priimek delavca, izobrazba delavca, šifra OZD)

projekt (šifra projekta, ime projekta, sredstva

udeleženec (matična št. delavca, šifra projekta naloga delavca)

V tem primeru je vsak objekt predstavljen z eno tabelo.

2. KREIRANJE PODATKOVNE BAZE IN REKONSTRUKCIJA

Elemente baze podatkov definiramo z interaktivnim VAX Rdb/VMS pomožnim programom (utility) RDO (Relational Database Operator). Definiramo naslednje elemente podatkovn baze:

- ime podatkovne baze in enote, na kateri se bo nahajala,
- relacije (z definicijo relacije je definiran zapis in v tem zapisu so vse komponente polja),
- polja (določimo imena, velikosti in tipe polj),
- objekte in atribute (vsak objekt mora imeti seznam atributov),
- vpoglede (views),
- omejitve (constraints)

in normaliziramo model. Proces normalizacije modela vsebuje eliminacijo dupliciranih podatkov in ponavljajočih se polj ter določitev ključev. Do vsakega atributa dostopamo preko ključa. Definicije atributov, objektov, relacij itd. se nahajajo v podatkovnem slovarju-CDD (Common Data Dictionary).

S stavkom DEFINE DATABASE poimenujemo bazo podatkov, kreiramo datoteke baze podatkov, kreiramo datoteko, v kateri se nahajajo začasni zapisi podatkovne baze (snapshort file).

Stavek

DEFINE DATABASE 'OVERNITE' IN 'CDD$TOP.BOOKKEEP
definira sledeče:

- podatkovni slovar
  CDD$TOP.BOOKKEEP
- datoteko s podatkovnimi elementi
  #BOOKKEEP#OVERNITE.RDB
- snapshort datoteko (datoteka z začasnimi zapisi)
  #BOOKKEEP#OVERNITE.SNB

Polja definiramo z DEFINE FIELD stavkom. Definicije polj se nahajajo v podatkovni bazi in CDD. Vsebujejo več atributov: globalne in lokalne.

Globalni atributi so:
- ime globalnega polja (imena polj se ne smejo ponavljati),
- kriteriji, katerim mora podatek ustrezati, preden se vpiše v polje,
- manjkajoče vrednosti (vrednost podatka je MISSING, če je vrednost polja neznana ali pa, če se ne sme uporabljati pri določeni aplikaciji).

Lokalni atributi so definirani z DEFINE RELATION stavkom.
Lokalni atributi so naslednji:
- ime lokalnega polja,
- DATATRIEVE pomožni stavki (support clause),
- COMPUTED BY stavek (s COMPUTED BY stavkom poimenujemo polje, ki je rezultat računske operacije).

Za vsako polje je treba določiti tip podatka. Preko seznama obstoječih imen polj lahko definiramo relacije. Relacije definiramo z uporabo RDO stavka DEFINE RELATION, s katerim imenujemo relacijo in napišemo seznam imen polj, ki jih vsebuje.

Za iskanje relacij za izbrane zapise se uporabljajo indeksi. Indeksi zapisov so shranjeni v posebni tabeli, ki je dodana Rdb/VMS podatkovni bazi. Rdb/VMS uporabi indeks, da najde zapis v bazi direktno.

Indeksi nam omogočajo spreminjanje, iskanje podatkov itd..

Vpogled (view) definiramo z RDO stavkom DEFINE VIEW, s katerim poimenujemo vpogled in napišemo seznam polj, ki jih vsebuje. Vpogled je virtualna relacija, ki kombinira polja iz ene ali več relacij.

Omejitve (constraints) se nanašajo na vrednosti polj v Rdb/VMS podatkovni bazi. Omejitev lahko definiramo z RDO stavkom DEFINE CONSTRAINT, v katerem poimenujemo omejitev in napišemo ime polja, na katerega se ta omejitev nanaša. Omejitve za polja se lahko postavi tudi v stavek DEFINE FIELD z uporabo klavzule VALID IF.

V podatkovno bazo vstavljamo podatke na tri načine:

- uporabimo Rdb/VMS interaktivni pomožni program RDO (s STORE stavkom dodajamo zapise),
- uporabimo DATATRIEVE za vpis že obstoječih podatkov v RMS datotekah v bazo podatkov;
- napišemo program v višje programskem jeziku, ki zapisuje vnešene zapise v datoteko ali pa prepisuje zapise iz RMS datotek v datoteko RDB podatkovne baze.

Že definirano podatkovno bazo lahko rekonstruiramo. Rekonstrukcija podatkovne baze zahteva precej dela in truda, zato je učinkovitejše, da začetnemu načrtovanju podatkovne baze posvetimo posebno pozornost. Za rekonstrukcijo se odločimo le, če je nujno potrebno. Možno je spreminjati relacije med podatki, brisati relacije, dodajati, brisati ali spreminjati elemente podatkovne baze. Brisanje relacij je enostavno le, če podatki, ki se nanašajo na relacijo, niso shranjeni v podatkovni bazi. Relacijo brišemo z ukazom DELETE ime relacije. Relacijo se ne sme brisati, če so druge relacije odvisne od te, ki jo želimo brisati. Rdb/VMS javi napako v primeru, ko želimo tako relacijo brisati. Če želimo polja dodajati, moramo napisati program, ki spremeni zapis podatkov v nov daljši zapis.

3. ZAŠČITA PODATKOV V PODATKOVNI BAZI

VAX Rdb/VMS poskrbi za zaščito podatkov v bazi pred uporabniki. Uporabniki, ki niso pooblaščeni za spreminjanje, brisanje, čitanje podatkov itd., tega ne morejo početi. VAX Rdb/VMS zaščita je odvisna od seznama za kontrolo dostopa-ACL (Access Control Lists). Ta seznam vsebuje naslednje podatke:

- indentifikacija uporabnika,
- katere operacije ta uporabnik lahko uporablja.

Z ukazom SHOW PROTECTION pogledamo vsebino ACL.

Pri vključevanju uporabnika v Rdb/VMS podatkovno bazo, Rdb/VMS bere ACL podatkovne baze in pregleduje, če je dan UIC v seznamu. Ko Rdb/VMS najde iskan UIC se iskanje konča. Rdb/VMS uporablja dva seznama za kontrolo dostopa, enega za dostop do podatkovne baze in drugega za dostop do relacij. Vstop uporabniku v podatkovno bazo omogočimo z DEFINE PROTECTION FOR DATABASE stavkom. Za vsako relacijo do katere ima dostop uporabnik ali grupa, uporabimo DEFINE PROTECTION FOR RELATION stavek.

4. PRIPOMOCKI ZA UPORABLJANJE PODATKOVNE BAZE

4.1. Podatkovni slovar

V podatkovnem slovarju se nahajajo definicije vseh podatkov, ki jih uporabljajo VAX ACMS aplikacije (definicije poslov itd.), VAX DATATRIEVE (opisi domen, zapisov, tabel, itd.), VAX Rdb/VMS (definicije polj, indeksov, omejitev, vpogledov itd.) in VAX TDMS (oblike, vprašalniki, knjižnjice z vprašalniki itd.).

Zgradba podatkovnega slovarja je hierarhična. Glavno vozlišče se imenuje CDD$TOP. Na to vozlišče lahko obesimo več naslednikov, ki so lahko direktoriji ali podslovarji (subdictionary) ali pa objekti. S pomožnim programom DMU (Data Manipulation Utility) kreiramo podslovarje, novo fizično datoteko (uporabimo ukaz CREATE), preimenujemo vozlišča (ukaz RENAME), gledamo po podatkovnem slovarju (ukaz LIST), brišemo podatke iz podatkovnega slovarja (uporabimo ukaz PURGE ali DELETE), kopiramo podatke iz določenega področja v neko drugo področje (ukaz COPY) in zaščitimo podatke pred uporabniki. Za določeno skupino uporabnikov ali pa za določenega uporabnika določimo:

- če uporabnik lahko briše podatke iz podatkovnega slovarja,
- če uporabnik lahko dodaja podatke v podatkovni slovar,
- če uporabnik lahko prehaja k naslednikom,
- če uporabnik lahko bere objekt iz podatkovnega slovarja,
- če uporabnik lahko dopolni zapis v podatkovnem slovarju,
- če uporabnik lahko kreira naslednike in podslovarje.

Podatke iz podatkovnega slovarja ali podslovarja lahko prepišemo v datoteko (ukaz BACKUP) ali pa restavriramo podatke iz te datoteke z ukazom RESTORE.

Vsak element RDB baze opišemo in ga shranimo v CDD. Za zapis različnih definicij v CDD uporabimo pomožni program CDDL (Command Data Dictionary Language). Definicija tvorjena s pomožnim programom CDDL se ne zapiše v podatkovi slovar v primeru, ko je napačna. Da hitreje najdemo napako, uporabimo pomožni program CDDV (Command Data Dictionary Verification), ki popravi napako, če jo zna popraviti ali pa pove, kje obstaja.

4.2. TDMS (Terminal Data Manipulation System)

VAX TDMS aplikacije uporabljamo za povpraševanje po podatkih in za oblikovanje menujev za vnos podatkov. Program z VAX TDMS aplikacijo izvaja serije rutin in prenaša podake v ali pa iz podatkovne baze. Te rutine so neodvisne od programa.

Vsaka TDMS aplikacija vsebuje naslednje elemente:

- aplikacijski program,
- eno ali več definicij zapisa,
- eno ali več definicij oblik (forms),
- eno ali več definicij vprašalnikov (requests),

- eno ali več definicij knjižnjic (library),
- eno ali več definicij datotek s knjižnjicami (file library).

Aplikacijski program

- bere ali pa izpisuje podatke iz podatkovne baze,
- kliče TDMS rutine,
  - za odpiranje in zapiranje datoteke s knjižnjico z vprašalniki (request library files) ter vhoda in izhoda na terminal,
  - za izvajanje vprašalnikov (requests),
  - za čitanje podatkov iz rezerviranih področij na terminalu,
  - za izpisovanje na terminal iz rezerviranih področij,
- javlja napake o procesu.

Aplikacijski programi napisani v programskih jezikih, ki komunicirajo s podatkovnim slovarjem (COBOL, FORTRAN, PASCAL, BASIC, PL/1) imajo možnost, da kličejo TDMS definicije, definicije zapisa itd. iz podatkovnega slovarja.

Definicija oblike določa obliko, ki se bo pojavila na terminal v času izvajanja programa. VAX TDMS obliko kreiramo direktno na zaslon terminala.
Oblika je definirana z naslednjim:

- s sliko na ekranu (določena so polja, v katera se vnaša ali pa izpisuje podatke in tekst, ki se izpisuje na ekran),
- z lego in dolžino polja,
- z nizom atributov za vsako polje (ti atributi so značilnosti polj ali pa pogoji, katerim mora ustrezati vnešen podatek),
- z obliko pomoči (HELP) pri vnosu.

VAX TDMS pomožni program za definicijo oblike-FDU (Form Definition Utility) pregleda sintakso definicije oblike in jo shrani v CDD, če v njej ne najde napake.

TDMS vprašalnik (request) je ključni element v TDMS aplikaciji. TDMS vprašalnik kontrolira pretok podatkov med zapisom in terminalom in med zapisom in podatkovno bazo. Kreira se z VAX TDMS pomožnim programom RDU (Request Definition Utility), ki shrani vprašalnik v CDD.

Seznam vprašalnikov se nahaja v knjižnjici z definicijami vprašalnikov (Request Library Definition). Knjižnjica z definicijami vprašalnikov se tudi kreira s pomožnim programom RDU (Request Definition Utility) in se shrani v CDD. Kreira se datoteka tipa RLB, ki vsebuje naslednje:

- imena vprašalnikov v knižnjici z definicijami vprašalnikov,
- definicije oblik v vprašalniku,
- informacije o zapisu, ki so v vprašalniku.

Če želimo spremeniti vprašalnik, obliko ali definicijo zapisa potem, ko je bila RLB datoteka že tvorjena, je treba RLB datoteko ponovno tvoriti.

Ko se izvaja program za vnos s TDMS aplikacijo, TDMS prikaže obliko na zaslon. Na označena polja vpisujemo podatke, ki se nato prenašajo v zapis podatkovne baze. Program za izpis podatkov v obliko pa prikaže na zaslon obliko in podatke iz iskanega zapisa podatkovne baze vpiše na označena polja.

4.3. VAX DATATRIEVE

VAX DATATRIEVE je programsko orodje, ki nam omogoča:

- izpis podatkov, ki so shranjeni v VAX RMS datotekah in VAX Rdb/VMS podatkovni bazi (podatke lahko predstavimo tudi grafično),
- iskanje določenih podatkov (uporabimo ukaz FIND),
- sortiranje podatkov na različne načine,
- izpis poročila o podatkih v željeni obliki,
- shranjevanje podatkov (ukaz STORE),
- spreminjanje podatkov,
- brisanje podatkov,
- definiranje novih relacij med zapisi in datotekami.

VAX DATATRIEVE izvaja tudi aritmetične operacije in izpisuje rezultate na terminal ali v datoteko.

Podatki, ki jih VAX-11 DATATRIEVE uporablja so shranjeni v domenah (domain). Domene povezujejo podatke v datotekah z definicijo zapisa. Domena je definirana z imenom domene, imenom datoteke s podatki in z definicijo zapisa. DATATRIEVE nam omogoča, da shranimo različne definicije o zapisih, objektih itd. v CDD in da si lahko vsak posameznik definira domene. Domene definiramo z ADT (Application Design Tool), ki vprašuje po informacijah, ki so potrebne za kreiranje domene, zapisa in datoteke s podatki. ADT kreira datoteko s podatki, tako kot je definirano z zapisom. Informacije, ki jih rabimo za shranjevanje podatkov z ADT so sledeče: ime domene, določitev datoteke s podatki, imena polj v zapisu, tip podatkov v vsakem polju, format polja, dolžina polja, organizacija podatkov, ime in atributi vsakega kjuča pri indeksni datoteki in ime komandne procedure, ki vsebuje definicije.

Če imamo zapise že shranjene v VAX RMS ali pa v VAX Rdb/VMS datotekah lahko iz teh datotek izberemo le določene zapise. Uporabimo RSE (Record Selection Expresion). Z RSE definiramo pogoje, ki jih določen zapis mora imeti, da ga lahko uvrstimo med izbrane zapise.

Primer RSE:

```
PRINT
1____FIRST 3
2_____________PERSONNEL
3_____________________WITH SUP_ID=00055
4_____________________________SORTED BY L_NAME
```

1- DTR povemo, koliko izbranih zapisov hočemo
2- pove izvor zapisov
3- pove, katere zapise naj vključi
4- pove, kako naj bodo zapisi urejeni

4.4. ACMS (Application Control and Management System)

Če je bil program napisan v višje programskem jeziku in če je prišlo do sprememb (vrstni red poslov se je spremenil, spremenil se je zapis itd.) je bilo treba program ponovno napisati. VAX ACMS pa nam omogoča, da definiramo individualen posel, ki je ločen od aplikacijskega programa in je shranjen v CDD. VAX ACMS posli kličejo VAX TDMS vprašalnike (requests) in programe.

VAX ACMS aplikacije so sestavljene iz naslednjih elementov:

- eno ali več grup poslov,
- enega ali več menujev s seznami poslov, ki jih je mogoče izbrati,
- definicije aplikacij, ki opisujejo posle v aplikaciji.

Vsak posel (task) se deli na več korakov (steps). Vsak posel je definiran na naslednji način:
- v editorju kreiramo datoteko z ADU ukazi,
- izvedemo komandno proceduro, ki javi napake, če obstajajo.

Če napak ni, se definicija posla shrani v CDD. Uporabimo ADU ukaz CREATE ali REPLACE. REPLACE ukaz deluje enako kot CREATE, če definicija posla še ne obstaja v CDD. Če defininicija posla ni pravilna prvič, potem moramo uporabljati REPLACE ukaz. Posli v ACMS aplikacijah morajo biti kombinirani v eno ali več grup poslov. Ko definiramo grupo poslov moramo poimenovati posle, ki pripadajo grupi. Če uporabljamo datoteko, v kateri je knjižnjica z vprašalniki, je treba to ime navesti.

Tipične VAX ACMS aplikacije omogočajo dostop do skupne podatkovne baze večim uporabnikom z malo ali pa brez izkušenj v računalništvu. Uporabnik vstopi v ACMS aplikacijo preko menuja. V menuju obstaja seznam poslov, ki jih vsebuje neka aplikacija. Ko uporabnik izbere posel, ACMS uporabi definicijo posla za določitev vrstnega reda, v katerem se bodo klicali vprašalniki in programi.

Primer ACMS menuja:

ACMS
MENU ZA UPRAVLJANJE APLIKACIJE

1	Dodajanje
2	Spreminjanje
3	Izpis na terminal
4	Editor
5	Pošta
6	DATATRIEVE

Izbira:

Posle, grupe poslov in menuje definiramo z VAX ACMS pomožnim programom ADU (Application Definition Utility).

Za izpis oblike menuja uporabljamo programsko orodje TDMS.

5. ZAKLJUČEK

Članek je opisal različna programska orodja, ki bodo omogočala tekoče vodenje podatkov ter hitrejše obdelovanje in izkazovanje podatkov na Zavodu SR Slovenije za družbeno planiranje. Posamezna programska orodja se med seboj dopolnjujejo. Nekatere operacije nad podatkovno bazo pa je mogoče izvesti z večimi programskimi orodji. Npr. zapise podatkovne baze tekoče vodimo in spreminjamo z interaktivnim programom RDO, z DATARIEVE ali pa s programom napisanim v višjem programskem jeziku, ki komunicira s CDD (FORTRAN, COBOL, BASIC, PASCAL).

6. LITERATURA

/1/ Suad Alagic, Relacione baze podataka. Svjetlost, Sarajevo 1984

/2/ Lojz Suc, Načrtovanje baze podatkov. Izobraževalni center DELTA, Ljubljana 1982

/3/ VAX Rdb/VMS Guide to Database Design and Definition. Digital Equipment Corporation, 1984

/4/ VAX Rdb/VMS Guide to Programing. Digital Equipment Corporation, 1984

/5/ Introduction to VAX-11 DATATRIEVE. Digital Equipment Corporation, 1984

/6/ Using VAX Information Arhitecture. Digital Equipment Corporation,1985

/7/ Marjan Krisper in Franc Žerdin, Informacijsko inženirstvo. Fakulteta za elektrotehniko v Ljubljani, 1985

/8/ Introduction to Application Development with the VAX Information Arhitecture. Digital Equipment Corporation, december 1985

UDK 681.324

Branko Mihovilovič, Peter Kolbezen, Jurij Šilc
Institut »Jožef Stefan«, Ljubljana

Komunikacijske povezave med transputerji predstavljajo zaradi cenenosti, enostavnosti in preproste uporabe nov standard v povezovanju računalnikov. V prispevku so na kratko opisani occamski konstrukti, ki so osnova pri načrtovanju transputerskih sistemov. Na primeru smo pokazali kako pomembno vlogo igrajo komunikacijski procesi pri načrtovanju transputerskih sistemov, ter kako pomembna je njihova preureditev v programu, če želimo varno in v polni meri izkoristiti sočasno izvrševanje opravil v sistemu.

Communicating processes in transputer systems - The transputer communication link is a new standard for computer system interconnection, since it is a cheap, simple and easy-to-use. This paper describes the occam constructs that are fundamental for transputer systems design. Their use is given by an example where some program transformations are performed for the sake of achieving the advantages of concurrent operating transpur system.

1. Uvod

Transputerji oziroma transputerski sistemi temeljijo na krmilno vodenem računanju in jih uvrščamo v skupino arhitektur, katerih model računanja je bodisi zaporedni ali sočasni krmilni tok [1]. Ti sistemi dovolj pogumno in verjetno tudi uspešno začrtujejo novo smer v razvoju računalnikov pete generacije. Organizacija stroja je sicer centralizirana, toda izrabljajoč VLSI tehnologijo je mikro računalnik s pomnilnikom, procesorjem in komunikacijskim delom narejen kot ena komponenta (čip). S preprostim povezovanjem takšnih komponent pa je možno zgraditi visoko sposobne računalniške sisteme.

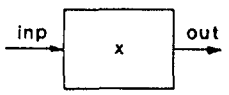
Vsporedno z razvojem materialne opreme, se je pojavila potreba po visokem programskem jeziku, ki pa ne bi bil namenjen samo programiranju računalniškega sistema, temveč bi bil primeren tudi za njegovo načrtovanje. Takšne zahteve danes izpolnjuje samo jezik OCCAM. Jezik je enostaven, sloni pa na dveh konceptih: sočasnosti in komunikaciji. Uporablja se enako dobro za opisovanje tako sistema medseboj povezanih mikroračunalnikov, kot za programiranje enega mikroračunalnika.

Semantično gledano, imamo na voljo množico pravil, ki nam služijo pri ti. transformaciji, oziroma optimizaciji programa. Pri slednji načrtovalec eksperimentalno izbira med različnimi implementacijami programa, ki so enakovredne. S tem doseže najboljše lastnosti načrtovanega sistema v pogledu sočasnosti, velikosti sistema in podobno.

V OCCAM modelu je proces zaporedje akcij, ki jih imenujemo osnovni procesi. Ločimo tri osnovne procese. Vhodni in izhodni proces skrbita za prenos vrednosti spremenljivk preko occamskih kanalov (c ? x, c ! x), s pomočjo prireditvenega procesa pa spreminjamo vrednosti spremenljivk (v := e). Osnovne procese kombiniramo tako, da se le-ti v konstruktih lahko izvajajo sekvenčno, paralelno ali pa se v množici procesov izvaja tisti proces, ki je prvi pripravljen.

2. Sekvenčni konstrukt

Element



opravlja dva sekvenčna procesa - vhodni in izhodni. Konstrukt

P = (μX.inp ? x -> out ! x -> X)

zagotavlja, da se bo vhodni proces končal prej, ko bo začel izhodni proces. V occamu zapisan program ima naslednjo obliko:

```
WHILE TRUE
VAR x:
SEQ
  inp ? x
  out ! x
```

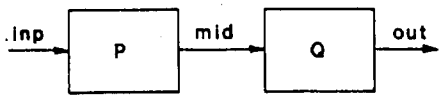
Tu velja opozoriti, da imamo v occamu na voljo **WHILE** stavek za opisovanje ponavljajočih procesov. To je povsem razumljivo, če vemo, da GO TO in podobni stavki ne sodijo v družino stavkov s katerimi popisujemo paralelne procese.

3. Paralelni konstrukt

Paralelni konstrukt

$P = (\mu X.inp\ ?\ x \rightarrow mid\ !\ (a*x) \rightarrow X)$
$Q = (\mu Y.mid\ ?\ y \rightarrow out\ !\ (b*y) \rightarrow Y)$
$R = P \parallel Q$

/ponazorjen z naslednjim elementom



povezuje sekvenčne procese. Vsak proces uporablja svoje spremenljivke, med seboj pa procesa komunicirata preko komunikacijskega kanala. V occamu zapisan program ima naslednjo obliko:

```
CHAN mid:
PAR
  WHILE TRUE
  VAR x, a, r:
  SEQ
    inp ? x
    r := a*x
    mid ! r
  WHILE TRUE
  VAR y, b, p:
  SEQ
    mid ? y
    p := b*y
    out ! p
```

Vobče sta procesa P in Q paralelna, ni pa nujno, da tečeta sočasno. Sočasen proces uporablja svoje spremenljivke, katerih ne more dodeljevati drugemu sočasnemu procesu. Sočasna procesa lahko komunicirata samo preko kanalov. Po enem kanalu poteka samo ena enosmerna, sinhronizirana komunikacija med dvema procesoma. V kanalski komunikaciji ne poznamo vmesnih pomnilnikov (bufferiranja).

4. Sočasni procesi

Bodita dana procesa P in Q, ki imata obliko:

$P = (\mu X.inp\ ?\ x \rightarrow outp\ !\ x \rightarrow X)$
$Q = (\mu Y.inp\ ?\ y \rightarrow outp\ !\ y \rightarrow Y).$

Poglejmo če obstaja možnost sočasnega izvajanja teh dveh procesov. Ta možnost je dana, saj lahko vhodni in izhodni komunikacijski proces tečeta sočasno: inp? x in outp! y, ter outp! x in inp? y. Vendar po pravilih o sočasnih procesih obstaja med sočasnima procesoma potencialno dead lock stanje, kar pomeni, da bi tudi procesa P in Q zašla v to stanje. Da preprečimo pojav dead lock stanja, moramo vsaj v enem paru sočasnih procesov en notranji proces začeti izvrševati predno se začne časovno prekrivanje procesnega para (premaknitev procesa po levi strani časovne osi). Zapišimo to še enkrat:

$(a \rightarrow P) \underset{c}{\parallel} (b \rightarrow Q) = (dead\text{-}lock)$

$(a \rightarrow P) \underset{c}{\parallel} (c \rightarrow d \rightarrow Q) = c \rightarrow ((a \rightarrow P) \underset{c}{\parallel} (d \rightarrow Q)).$

Zgornji izrazi nam povedo, da lahko nek proces delimo (če je to mogoče) na podprocese, ki (če so med seboj povezani s kanali) tečejo sočasno z ostalimi podprocesi (tudi s tistimi, ki pripadajo drugemu procesu). Temu pravimo tudi dekompozicija procesa. Želimo, da je dekompozicija hierarhična (varovanje pred dead lock stanji), da omogoča uporabo paralelnih konstruktov in da so pri delitvi tisti končni procesi kar se da preprosti. Tako lahko ustvarimo programe, ki uporabljajo visoko stopnjo sočasnosti; vsak enostaven proces pa se izvaja na lastnem procesorju, ki je lahko enostaven zato pa izredno hiter (RISC).

V sekvenčnem procesu W

$W = (\mu X.inp?x \rightarrow out!x \rightarrow inp?y \rightarrow out!y \rightarrow X)$

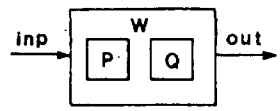
poiščemo tiste podprocese, ki se lahko izvršujejo sočasno (dekompozicija). Najdemo dva para sočasnih procesov

$P = P1 \underset{c}{\parallel} P2 = (inp\ ?\ x \rightarrow out\ !\ y \rightarrow P)$

$Q = Q1 \underset{c}{\parallel} Q2 = (out\ !\ y \rightarrow inp\ ?\ x \rightarrow Q)$

in dobimo

$W = inp\ ?\ y \rightarrow (\mu X.P \rightarrow Q \rightarrow X)$



5. Primer

Oglejmo si iterativno polje (kvadratna mreža transputerjev), ki je načrtano tako, da z njim lahko izračunamo m skalarnih produktov vektorjev v in w_i, $1 \le i \le m$. Na primeru bomo razložili nekatere posebnosti komunikacij med procesi oziroma transputerji v takšnem in podobnih sistemih. Naš primer zapišemo z naslednjim izrazom:

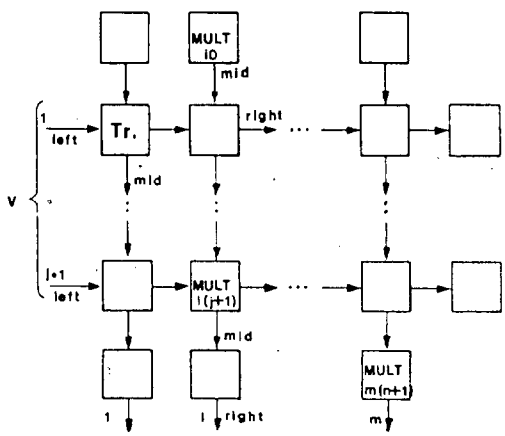
$$s_i = \sum_{j=0}^{n-1} w_{ij} \times v_j \ .$$

Primer torej rešimo z iterativnim poljem transputerjev, ki ga ponazarja slika 1. Vidimo, da gre v bistvu za izračun i skalarnih produktov med enim vhodnim vektorjem v in m vektorji w_i, katerih komponente (matrika W) so zapisane v posameznih transputerjih. Zapišimo sekvenco procesov, ki jih izvajajo posamezni transputerji v iterativnem polju:

$MULT_{10} = (\mu X.P_0 \rightarrow X)$

$MULT_{i(j+1)} = (\mu X.P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4 \rightarrow P_5 \rightarrow X)$

$MULT_{m(n+1)} = (\mu X.P_6 \rightarrow P_7 \rightarrow X)$



slika 1

kjer so

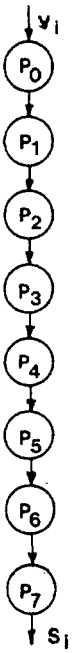
$P_0 = mid_0 ! 0$ $\quad P_1 = left_i ? v$

$P_2 = mid_{ij} ? y$ $\quad P_3 = z := (w_{ij} * v) + y$

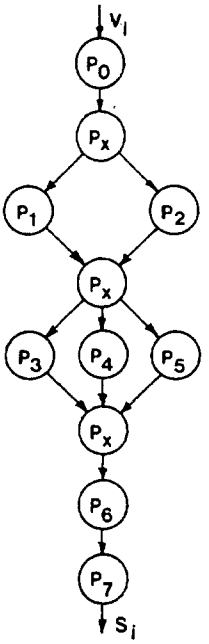
$P_4 = mid_{i(j+1)} ! z$ $\quad P_5 = right_j ! v$

$P_6 = mid_{mn} ? x$ $\quad P_7 = right_{i(n+1)} ! s$

Na sliki 2a je podan preprost grafni model programa za eno transputersko celico. Postavlja se vprašanje, kako v najboljši meri izkoristiti prednosti iterativnih polj, tj. možnosti pohitritev pri reševanju takšnih in podobnih problemov. Ali povedano drugače, zagotoviti, da kar največ celic lahko deluje sočasno.



slika 2a



slika 2b

V našem primeru najdemo dve skupini procesov, ki se lahko izvršujejo sočasno. V prvi skupini sta procesa P1 in P2, v drugi pa so procesi P3, P4 in P5. Grafni model na sliki 2b prikazuje takšno strukturo sočasnih procesov. Procesi Px združujejo v sebi vse potrebne nadzorne strukture paralelnih oziroma sočasnih procesov (sinhronizacija). V occamu zapisan program za n x m celic ima sedaj naslednjo obliko:

```
PROC MULT (CHAN mid, left, right)
  VAR y, v, m, z
  SEQ i = [1 FOR m]
    SEQ
      PAR
        left ? v
        mid ? y
      PAR
        z := W * v + y
        mid ! z
        right ! v
```

Z resnejšim premislekom ob sliki 2b in zgornjem programu ugotovimo, da lahko le v n transputerjih tečejo procesi sočasno in da je zakasnitev pri izvajanju procesa P5 kriva za to, da procesi v ostalih m-1 verigah transputerjev ne tečejo sočasno (m skalarnih produktov). Slednje bi bilo možno ob pogoju, da procesa P1 in P5 tečeta sočasno, vendar na enem transputerju takšna procesa ne moreta teči sočasno. Rešitev je v časovnem zamiku med sekvenčnima procesoma vsaj za en obhod zanke. Takšno transformacijo bi zapisali v naslednji obliki:

```
SEQ i=[0 FOR m]          SEQ
  SEQ                      P(0)
    P(i)          =>       SEQ i=[0 FOR m-1]
    Q(i)                     SEQ
                               Q(i)
                               P(i+1)
                           Q(m)
```

Occamski program z zgornjo transformacijo dobi naslednjo obliko:

```
PROC MULT (CHAN mid, left, right)
  VAR y, v, m, z
  SEQ
    left ? v
    mid ? y
    SEQ i = [ 1 FOR m - 1 ]
      SEQ
        PAR
          z := M * v + y
          mid ! z
          right ! v
        PAR
          left ? v
          mid ? y
    PAR
      z := m * v + y
      mid ! z
      right ! v
```

Hkrati s transformacijo smo rešili tudi problem pojava dead-lock stanja pri sočasnem izvajanju procesov P4 in P5. Če pogledamo zančni del zgornjega programa ugotovimo, da imamo zaporedje dveh izhodnih in dveh vhodnih komunikacijskih procesov. V takšnem zaporedju sočasnih procesov kaj hitro pride do sinhronizacijskih težav in s tem do dead-lock stanja. Težavam se izognemo tako, da spremenimo ime spremenljivke vhodnima komunikacijskima procesoma in dodamo prireditvena procesa.

```
---                          ---
PAR                          PAR
  left ? v        =>           SEQ
  mid ? y                        left ? a
---                              v := a
                               SEQ
                                 mid ? b
                                 y := b
                             ---
```

Prireditvena procesa želimo ločiti od komunikacijskih procesov. To bomo storili z naslednjo transformacijo. Označimo s C1 in C2 komunikacijska procesa, s P in Q pa prireditvena procesa. Glede na to, da procesa C1 in C2 ne začneta istočasno lahko zapišemo:

$(C1 \rightarrow P) \underset{C}{\|} (C2 \rightarrow Q) = ((C1 \rightarrow P) \underset{C}{\|} C2) \rightarrow Q =$

$(C2 \underset{C}{\|} (C1 \rightarrow P)) \rightarrow Q = ((C2 \underset{C}{\|} C1) \rightarrow P) \rightarrow Q =$

$(C2 \underset{C}{\|} C1) \rightarrow P \rightarrow Q = (C1 \underset{C}{\|} C2) \rightarrow P \rightarrow Q$

Če upoštevamo zgornjo transformacijo, dobi naš program končno obliko:

```
PROC MULT (CHAN mid, left, right)
  VAR y, v, m, z, a, b
    SEQ
      left ? v
      mid ? y
      SEQ i = [1 FOR m - 1]
        SEQ
          PAR
            z := m * v + y
            mid ! z
            right ! v
          PAR
            left ? a
            mid ? b
        v := a
        y := b
      PAR
        z := m * v + y
        mid ! z
        right ! v
```

Procesiranje podatkov v zadnjih dveh programih je popolnoma enako, razlika pa je v tem, da v zadnjem programu ne more priti do izpisov rezultata (mid ! z), predno se ne izvedeta vhodna sočasna procesa (left ? a, mid ? b).

6. Zaključek

Nedvomno je jezik occam v primerjavi s sorodnimi jeziki najprimernejši tako za opisovanje strukture, kot za programiranje transputerskega sistema in njegovih komponent. Na primeru smo pokazali pomen podrobnejše analize programskega dela transputerskega sistema; posledica neustrezne časovne razporeditve procesov je lahko neučinkovitost celotnega sistema. Komunikacijski procesi igrajo pomembno vlogo pri izbiri tistih procesov, ki jih lahko združimo v sisteme sočasnih procesov. Pokazali smo, da lahko sočasnosti formiramo na dveh nivojih:

a) kjer več transputerjev deluje sočasno, ter

b) kjer na enem transputerju sočasno poteka več procesov.

7. Literatura

[1] Borut Robič, Jurij Šilc, Razvrstitev novogeneracijskih računalniških arhitektur, Informatica **10** (4) (1986) 18-32.

[2] Branko Mihovilović, Slavko Mavrič, Peter Kolbezen, Transputer - osnovni gradnik večprocesorskih sistemov, Informatica **10** (4) (1986) 81-84.

[3] C.A.R. Hoare, Communicating sequential processes, Prentice-Hall International (1985).

[4] Jane Curry, Occam solves classical operating system problems, Microprocessors and Microsystems **8** (6) (1984) 280-283.

[5] David May, Richard Taylor, Occam - an overview, Microprocessors and Microsystems **8** (2) (1984) 73-79.

[6] Richard Taylor, 'Transputer communication link, Microprocessor and Microsystems **10** (4) (1986) 211-215.

[7] David May, Roger Shepherd, The transputer implementation of occam, Proc. Int'l Conf. Fifth Generation Computer Systems 1984, OHM North-Holland (1984) 533-541.

PROGRAMIRANJE Z MODULO – 2
1. del

A. Brodnik, M. Špegel, T. Lasbaher
Institut »Jožef Stefan«, Ljubljana

UDK 681.3.06:519.682 MODULO-2

V članku opisujemo programski jezik modula-2. Opis je razdeljen na dva dela. V prvem obravnavamo osnovne podatkovne, ukazne (stavki) in programske (moduli, podprogrami) strukture. Tako je podrobno opisan pojem mudula in z njim povezanega modularnega programiranja. Poleg tega opisujemo tudi osnovna načela sistemskega oziroma nizkonivojskega programiranja v moduli-2. Nadaljevanje članka je namenjeno primerjavi Module-2 s tremi drugimi sodobnimi jeziki, in sicer s pascalom, ado in z jezikom c.

Programming With Modula-2 In this paper is a description of the Modula-2 programming language. The description is divided into two parts. In the first part, we discuss the essential data, control, and programming structures of the language which separate Modula-2 from other modern languages, concluding wise the notation of a module and modular programming, and some principles of system or low-level programming. In the second part of this contribution, Modula-2 is compared to other three modern programming languages: Pascal, Ada and C.

1. UVOD

V članku želimo v dveh delih predstaviti nov, moderen programski jezik. Za celovit opis jezika pa je okvir tega članka in njegovega nadaljevanja preozek, poleg tega pa je bralcu na voljo vrsta dobrih knjig [3, 7, 8, 14, 18]. Pač pa je namen našega prispevka z opisom bistvenih novosti in programirnih možnosti vzpodbuditi bralca k poglobljenemu spoznavanju tega nedvomno zelo sposobnega in morda enega temeljnih jezikov naslednjih let [2].

Programski jezik modula-2 je neposredni naslednik module-1, oba pa izvirata iz pascala. Osnovnim pravilom so bili dodani predvsem nekateri elementi paralelnega pascala ("Concurrent Pascal")[5] ter možnost modularnega programiranja. Zato osnovo jezika tvorijo že dobro poznani stavki iz pascala [6, 9], ki pa so v moduli-2 še bolj poenostavljeni. S tem je jezik postal preglednejši, programi pa bistveno bolj zgoščeni. Modula-2 ni boljša od pascala samo v pogledu strukturiranosti, ampak omogoča z določenimi ukaznimi strukturami tudi zelo učinkovito programiranje na sistemskem nivoju.

Izkušeni programerji, ki sicer uporabljajo katero od bolj izpopolnjenih različic pascala [15, 16], bodo dejali, da vse to, kar sicer prinaša modula-2, oni lahko že precej časa uporabljajo. To je res, vendar se moramo hkrati zavedati, da so to le dialekti pascala, medtem ko so vsi deli jezika, ki jih bomo tu opisali, že v osnovni definiciji module-2. Opis jezika je osnovna tema drugega poglavja. V tretjem poglavju, bomo prikazali nekatera načela modularnega programiranja, četrto poglavje pa je namenjeno prikazu zmožnosti module-2 za sistemsko ali nizkonivojsko programiranje.

V drugem delu tega članka bomo primerjali jezik s pascalom, ado in jezikom c. Pri tem se bomo seznanili z nekaterimi prav zanimivimi spoznanji. Videli bomo na primer, da ima modula-2 pravzaprav skoraj vse bistvene prednosti, ki jih programerji sicer pripisujejo adi in c-ju.

V drugem poglavju nadaljevanja so zbrani še primeri nekaterih prevajalnikov za različne operacijske sisteme in zanimiv primer povezave module-2 s prologom.

V dodatku sta priložena kratka primera, ki naj predočita možnosti modularnega in sistemskega programiranja v moduli-2.

2. Opis jezika modula-2

Pri razmišljanju o uporabi kakšnega programskega jezika, moramo vzeti v pretres predvsem tri gradnike: podatkovne strukture, ukazne strukture (stavke) in programske strukture. Opis jezika smo zato razdelili nekako na tri dele.

Zaradi smotrne zgradbe jezika so programi napisani v moduli-2 [18] zelo pregledni. Jezik dobro podpira strukturiranje programov, in sicer tako v pogledu podatkovnih kot ukaznih struktur. Razen tega modula-2 loči med velikimi in malimi črkami v imenih spremenljivk. Preglednost programov povečuje še zahteva, da so vse rezervirane besede napisane z velikimi črkami.

Pri opisu jezika predpostavljamo določeno stopnjo znanja katerega od algolskih jezikov, a najbolje je poznavanje pascala.

2.1 Podatkovne strukture

Vsak programski jezik pozna nekaj osnovnih tipov. Najbolj običajni osnovni tipi so **integer** (cela števila), **real** (realna števila) in **boolean** (logične spremenljivke) in jih najdemo v večini jezikov od fortrana [12] pa do ade [13, 17]. V večini jezikov algolskega tipa, kamor sodi tudi modula-2, poznamo še tipe **char** (črkovna spremenljivka), **set** (množica) in **pointer** (kazalec).

V modulo-2 so vgrajeni vsi zgoraj navedeni tipi, poleg njih pa pozna jezik še nekaj tipov, ki so uporabni predvsem pri programiranju na

nižjem nivoju. To so tipi **CARDINAL** (nenegativna števila), **WORD** (beseda), **ADDRESS** (naslov), **BITSET** (množica bitov) in **PROC** (podprogram). Vlogo teh tipov si bomo ogledali v četrtem poglavju tega prispevka.

2.1.1 Sestavljeni tipi

Podobno kot pascal pozna tudi modula-2 sestavljene podatkovne tipe. To so zapisi (RECORD), zapisi z različicami, polja (ARRAY), naštevalni tipi in delni tipi. Oglejmo si oblike definicij za omenjene podatkovne tipe na naslednjem primeru:

```
TYPE
  zapis  = RECORD part1, part2: CARDINAL END;
  zapis1 = RECORD
             CASE selekt: BOOLEAN OF
               TRUE:  ch: CHAR  |
               FALSE: ca: CARDINAL
             END
           END;
  polje  = ARRAY [1..10], [1..10] OF INTEGER;
  nastej = (da, ne, mogoce);
  delni  = [0..9];
```

2.1.2 Pretvarjanje med tipi

V moduli-2 je obravnavanje spremenljivk močno vezano na njihov tip. Ne moremo na primer, seštevati celega in realnega števila kot v pascalu. Prepovedano je celo seštevanje celega in nenegativnega števila. Za premostitev teh problemov obstaja cela vrsta funkcij, ki samo na logičnem nivoju pretvorijo tip spremenljivke, na velikost izhodne kode pa nimajo vpliva. Imena teh funkcij so enaka imenom tipov. Tako lahko naredimo naslednjo pretvorbo:

```
MODULE Primer1;
VAR c: CARDINAL;
    i: INTEGER;
BEGIN
  i := INTEGER (c); c := CARDINAL (i)
END Primer1.
```

Seveda je pretvarjanje omejeno le med tipi z enako dolžino. Kot primer pravilne pretvorbe, si oglejmo naslednji podprogram:

```
PROCEDURE Prepisi (iz: IzTip;
                   VAR v: vTip;
                   VAR prepisano: BOOLEAN);
BEGIN
  prepisano := TSIZE(IzTip) = TSIZE(vTip);
  IF prepisano THEN v := vTip (iz) END
END Prepisi;
```

Klicani podprogram TSIZE je eden redkih, ki so definirani v sistemu module-2 in izračuna velikost tipa, ki je parameter podprograma, v besedah.

2.2 Ukazne strukture

Modula-2 pozna naslednje stavke:
- prirejanje
- klic podprograma
- vrnitev iz podprograma (RETURN)
- pogojni stavek (IF, ELSIF, ELSE)
- izbiralni stavek (CASE)
- izhodni stavek (EXIT)
- strukturni izbirni stavek (WITH)
- zanke
  - nedoločna (LOOP)
  - pogoj na začetku (WHILE)
  - pogoj na koncu (REPEAT)
  - naštevalna zanka (FOR)

Modula-2 torej ne uvaja nobenih bistveno novih stavkov, pač pa je le njihova oblika lepše zapisana. Na primer, modula-2 ne pozna sestavljenega stavka (BEGIN ... END), ampak se vsi njeni stavki razen prirejanja, vrnitvenega in izhodnega končajo z rezervirano besedo END. V primerjavi s pascalom lahko opazimo odvečnost besedice **begin** na naslednjem primeru:

```
MODULE Primer;          program Primer;
BEGIN                   begin
  IF pogoj THEN           if pogoj then begin
    stavek1;                stavek1;
    stavek2                 stavke2
  END                     end
END Primer.             end.
```

Novo je tudi ločevanje posameznih možnosti v izbiralnem stavku. Ta ima sedaj obliko:

```
CASE izraz OF
  moznost1 : Stavki1 |
  moznost2 : Stavki2
  ELSE Stavki3
END;
```

2.3 Moduli in podprogrami

Osnovna programska enota v jeziku ni več podprogram, temveč modul. Ta lahko združuje več podprogramov in modulov. Modul in podprogram se razlikujeta tudi v svoji zasnovi, saj je modul le logična, programska enota, ki obstaja takorekoč samo na nivoju izvorne kode in je njegova vloga samo povečanje preglednosti programa. Po drugi strani pa podprogram predstavlja osnovno izvajalno enoto in obstaja tako na nivoju izvorne kot tudi izvajane kode.

2.3.1 Moduli

Modula-2 poznam tri tipe modulov:
- programski modul
- definicijski modul (DEFINITION)
- delovni modul (IMPLEMENTATION)

Prvi služi kot osnovni uporabnikov program ali kot ga poznamo v pascalu **program**. Druga dva tipa modulov sta dodana predvsem zaradi zaokrožitve sistema jezika, s čimer se odpravijo različne uporabniške knjižnice. Modula-2 ne pozna več zunanjih podprogramov (**external**) ali vstavljanja celih kosov programa (**include**), ampak samo navedemo modul, iz katerega želimo uporabiti določeno spremenljivko (ta je lahko poljubnega tipa, tudi PROC!).

Uporabnost definicijskih in delovnih modulov je najbolj očitna pri snovanju velikih projektov, pri katerih vedno sodeluje več ljudi. Takšno programiranje zahteva razdelitev dela in določitev nekaterih skupnih točk - spremenljivk. V primeru razvoja z modulo-2 so spremenljivke opisane z definicijskimi moduli. V definicijskih modulih torej snovalci opišejo kaj bodo podprogrami delali, kakšne bodo posamezne spremenljivke in podobno. Nato se programerji lotijo vsak svojega dela, kar pomeni, da vsak izdela svoj delovni modul, v katerem se določi **kako** se bodo izvajali.

Opisani postopek izdelave projekta v osnovi omogoča zelo enostavno popravljanje programov, saj ni potrebno prevesti zaradi popraviti ničesar drugega kot modul, ki je napačen, seveda ni pa edini možen način uporabe ločenega prevajanja posameznih modulov. Sicer ločeno prevajanje omogočajo že običajne sistemske knjižnice, ki pri drugih jezikih obstajajo izven opisa jezika, medtem ko je pri moduli-2 vse vgrajeno v sistem jezika samega. Podrobneje si bomo ogledali načela modularnega programiranja v tretjem poglavju.

Naslednja posebnost modulov je veljavnost\spremenljivk. Oglejmo si, kako v moduli-2 dostopamo do spremenljivk. Osnovno programsko enoto -

modul si lahko predstavljamo kot zaprt prostor, v katerem nam je vse vidno, zato pa ne vidimo ničesar zunaj tega prostora. Če želimo videti kakšno stvar, ki je zunaj prostora, si jo moramo posebej dodefinirati: sprejeti jo moramo v modul. V moduli-2 to naredimo s stavkom IMPORT. Ne moremo pa sprejemati poljubnih spremenljivk, ampak samo tiste, ki so na voljo v zunanjem svetu. Da lahko neka spremenljivka obstaja v zunanjem svetu, jo moramo iz modula oddati. Za to služi stavek EXPORT. Torej si lahko vse oddane spremenljivke predstavljamo kot nekakšno prosto blago, ki ga lahko sprejme katerikoli modul.

To načelo modulov je tako močno, da tudi vsebina vsebovanih modulov ni poznana osnovnemu modulu. Da bo stvar jasnejša si poglejmo naslednji primer:

```
MODULE Ena;
  VAR x, y, z: CARDINAL
  MODULE Dva1;
    EXPORT a, b;
    VAR a, b, c: CARDINAL;
  BEGIN
    (* tu so dostopne
       spremenljivke a, b ali c *)
  END Dva1;

  MODULE Dva2;
    IMPORT x, a;
    VAR r, s, t: CARDINAL;
  BEGIN
    (* dostopne so spremenljivke
       x ali Ena.x, a ali Dva1.a,
       r, s, t *)
  END Dva2;
BEGIN
  (* tu pa spremenljivke x, y, z
     in Dva1.a ali a *)
END Ena.
```

2.3.2 Podprogrami

Poleg modulov modula-2 še vedno pozna tudi podprograme. Njihova definicija je podobna kot pri pascalu, adi ali drugih jezikih. Parametri podprogramov so lahko klicani po vrednosti ali po naslovu (referenci). V podprogramu nekega modula so vidne tudi vse zunanje spremenljivke, ki pa so definirane v tem modulu. Dostop do ostalih spremenljivk je možen preko oddajno/vnosnih struktur. Za lažjo predstavitev naj služi naslednji primer:

```
MODULE Glavni;
  VAR c: CARDINAL;
  MODULE Ena1;
    EXPORT a;
    VAR a: INTEGER;
  END Ena1;
  MODULE Ena2;
    IMPORT a, c;
    EXPORT Priredi;
    PROCEDURE Priredi;
    BEGIN
      a:=INTEGER(c)
    END Priredi;
  END Ena2;
BEGIN
  Priredi;
END Glavni.
```

V podprogramu lahko nastopa tudi vrnitveni stavek (RETURN), ki pomeni takojšen zaključek izvajanja podprograma. Če tega stavka ni, se vrnitev izvede, ko izvajanje podprogram pride do konca (END). V primeru funkcijskega podprograma sledi vrnitvenemu stavku izraz, ki določa vrednost funkcije.

Definicija funkcijskih podprogramov je pri moduli-2 in pascalu različna, saj jih pri pascalu označuje rezervirana beseda function. V moduli-2 funkcijo ločimo od podprograma le po tem, da je v glavi definicije naveden tip. Primer:

```
PROCEDURE PraviPodprogram;
BEGIN
END PraviPodprogram;

PROCEDURE Funkcija: tip;
BEGIN
END Funkcija;
```

Pri definiciji tipov podprogramskih parametrov obstajajo nekatere možnosti sproščanja sicer trdnega sovpadanja tipov. In sicer lahko uporabimo splošne tipe:

- WORD: osnovna računalniška beseda, sicer posplošenje kateregakoli tipa, ki je dolg eno besedo
- ADDRESS: kazalec na poljuben tip, sicer je definiran kot
  ADDRESS = POINTER TO WORD;
- BITSET: poljubna množica, sicer množica 16 bitov
- ARRAY OF tip: odprto polje poljubnega tipa, katerega velikost lahko določimo s funkcijo HIGH

Sistem jezika pozna zelo malo standardnih funkcij in podprogramov. S tem se je jezik izognil odvisnosti od določenega operacijskega sistema ali računalnika. Vsi standardni podprogrami izvirajo iz definicije jezika in omogočajo predvsem bolj parametrizirano pisanje programov. Seznam standardnih podprogramov si lahko bralec ogleda v knjigi [18].

Na tem mestu naj posebej poudarimo, da podprogrami za branje z različnih enot oziroma datotek ostajajo povsem uporabnikova skrb. Vendar proizvajalci prevajalnikov vedno dodajajo sistemu jezika nekakšno standardno množico modulov, ki omogočajo bralne in pisalne operacije na različne enote ter ostalo osnovno delo z operacijskim sistemom. Hkrati se v svetu vedno bolj čuti potreba po standardizaciji teh modulov. Določen naj bi bil predvsem vmesnik med modulo-2 in vhodno-izhodnimi funkcijami, ki so odvisne od posameznega sistema. Delovno ime tega projekta je OSSI in pomeni Operating System Standard Interface [1]. Izvaja se na večih mestih, njegovo središče pa je v Zurichu.

2.4 Procesi

Kot nasledstvo concurrent pascala so procesi. Proces predstavlja zaporedje ukazov, ki se izvajajo neprekinjno. Proces je tudi nekako osnovna fizično izvajana enota. Modula-2 omogoča uporabo sorutin (coroutines), kar pomeni, da nima vgrajenega razporejevalnika. Proces lahko poženemo s podprogramom TRANSFER, ki preda nadzor od trenutnega procesa naslednjemu, ki je naveden kot parameter. Podrobnejši opis tega mehanizma sledi v četrtem poglavju.

3. Modularno programiranje

Dandanes, ko cena strojne opreme vztrajno pada in se cena programske opreme dviguje, postajajo načela učinkovitega programiranja vse bolj pomembna. Prav modularno programiranje se je izkazalo za eno od temeljnih načel. V ZDA so v sedemdesetih letih sestavili celo ekipo strokovnjakov, ki naj predložila nov jezik, ki bo omogočal hitro in učinkovito, torej modularno programiranje. Nastala je ada. Vendar se je kmalu izkazalo, da je jezik prevelik, zaradi česar nikakor ni mogel dobiti širšega področja uporabe. Hkrati z njim se je pojavil manjši, vendar še vedno zelo učinkovit jezik - modula-2.

3.1 Delitev na module

Sedaj, ko smo dobili možnost uporabe ločenih modulov, jih moramo še nekako pametno uporabiti. Večina programerjev si vedno postavlja vprašanje: "Kaj sodi v en modul?".

Odgovor, ki bi bil plod globljega premisleka, bi se naslonil na dejstvo, da so pri izdelavi kakršnegakoli projekta podatki in njihove strukture navadno pomembnejše kot nadzorne strukture. Torej vežemo modul na pojem podatkovne strukture. Pomberger [11] sicer navaja še več načel pri delitvi programa na module, vendar na konou vseeno ugotavlja, da so podatkovne strukture najpomembnejši dejavnik, ki določa vsebino modula. Zmeraj pa to ni res. Kot protiprimer so moduli, ki upravljajo različne enote (driver, handler). V drugem delu članka bomo v enem od primerov predstavili takšen modul.

3.2 Kako so moduli povezani

Seveda mora modul vseeno še ohranjati stik z okolico. V ta namen nam služijo nekatere spremenljivke, predvsem pa različni podprogrami. Ker so podprogrami del modula, je s tem zagotovljena varnost podatkov pred napačno uporabo. V moduli-2 dosežemo povezavo z oddajo in vnosom določenih spremenljivk (EXPORT in IMPORT stavek). Naj zopet poudarimo, da je spremenljivka lahko poljubnega tipa in torej lahko tudi tipa PROC! Okolica seveda natančno pozna tipe oddanih spremenljivk in tudi parametrov v oddanih podprogramih že v času prevajanja. Prav močna zahteva po sovpadanju tipov je eno od osnovnih načel module-2. S tem so tudi odpravljene tako pogoste pomote, ki sicer rade nastopajo v drugih jezikih (fortran, pl/1 in drugi).

Za primer, kaj bi povezovalo nek modul z zunanjostjo, si oglejmo modul, ki hrani podatkovno strukturo sklada. Smiselno je oddati le podprograma Vstavi in Vzemi ter funkciji Prazen in Poln. To v moduli-2 definiramo na naslednji način:

```
DEFINITION MODULE NadzornikSklada;

  EXPORT Vstavi, Vzemi, Prazen, Poln;

  PROCEDURE Vstavi (el: CARDINAL);
  PROCEDURE Vzemi (VAR el: CARDINAL);
  PROCEDURE Prazen (): BOOLEAN;
  PROCEDURE Poln (): BOOLEAN;
END NadzornikSklada.
```

To je za nekoga, ki želi uporabljati sklad povsem dovolj. Kako programer deluje nad skladom, za njega ni važno.

3.3 Ločeno prevajanje

S tem, da smo razbili program na module, nismo le zaščitili posameznih podatkovnih struktur ali dostopov do različnih enot, ampak smo pridobili tudi možnost, da lahko vsak modul prevajamo povsem neodvisno od ostalih. Kot dokaj preprost a zelo pomemben primer uporabnosti ločenega prevajanja [4] je vzdrževanje programske opreme. Recimo, da se v obdobju izkoriščanja nekega programskega paketa izkaže, da je sklad, ki ga uporabljamo preko modula NadzornikSklada premajhen. Vse kar je treba spremeniti, je delovni modul, ga prevesti in ponovno vse skupaj povezati. Ostali moduli se pri tem prav nič ne spremenijo!

Naj na koncu poglavja navedemo še delovni modul NadzornikSklada, kot bi ga napisal povprečen programer. Če pa boste imeli probleme z velikostjo sklada, povečajte konstanto StackLimit. V primeru, da vam tudi to ni dovolj, lahko napišete nov modul, ki bo sklad hranil na datoteki. Vendar pozor! Ostali programi se tudi v tem primeru ne bodo prav nič spremenili!

```
IMPLEMENTATION MODULE NadzornikSklada;

  CONST VelikostSklada = 1000;
  VAR sklad: ARRAY [1..VelikostSklada]
               OF CARDINAL;
      n: CARDINAL;

  PROCEDURE Vstavi (el: CARDINAL);
  BEGIN
    IF n < VelikostSklada THEN
      n:=n+1; sklad[n] := el
    END
  END Vstavi;

  PROCEDURE Vzemi (VAR el: CARDINAL);
  BEGIN
    IF n > 0 THEN
      el := sklad[n]; n:=n-1
    END
  END Vzemi;

  PROCEDURE Prazen (): BOOLEAN;
  BEGIN
    RETURN n = 0
  END Prazen;

  PROCEDURE Poln (): BOOLEAN;
  BEGIN
    RETURN n = VelikostSklada
  END Poln;

BEGIN  (* inicializacija modula *)
  n := 0
END NadzornikSklada.
```

4. Sistemsko programiranje

Možnost sistemskega ali nizkonivojskega programiranja (low level programming) je ena bistvenih postavk, ki jih mora omogočati sodoben programski jezik. Vsekakor mora biti možen dostop do poljubnih naslovov v pomnilniku in pisanje prekinitvenih podprogramov. Prav slednja zahteva je močno povezana s pojmom procesa in še bolj s pojmom paralelnih procesov. Modula-2 je v pogledu zmožnosti opisovanja paralelnih procesov zelo močan jezik. Že v osnovi pozna pojem procesa. Procesi se izvajajo kot sorutine (coroutines), vendar lahko uporabnik ob ustrezni zunanji prekinitvi napiše razporejevalnik (schedular), ki omogoča navidezno sočasno izvajanje večih procesov. V drugem delu članka bomo med primeri prikazali prav takšen razporejevalnik.

4.1 Dostop do pomnilnika

Modula-2 omogoča dostop do poljubnega mesta v pomnilniku na dinamični ali statični način. Statični način (podobno kot pri OMSI pascalu [10]) pomeni, da ob definiciji spremenljivke navedemo tudi mesto, kjer se nahaja.

```
VAR NaNaslovu10 [10]: CARDINAL;
```

Dinamičen dostop pa je možen preko splošnega tipa ADDRESS, ki je definiran v modulu SYSTEM in ima obliko:

```
TYPE ADDRESS = POINTER TO WORD;
```

Tako lahko uporabljamo opisano mesto v pomnilniku tudi na sledeč način:

```
MODULE Primer2;
  FROM SYSTEM IMPORT ADDRESS;
  CONST xxx = 0;
  VAR a: ADDRESS;
BEGIN
  a := ADDRESS (10);
  a^ := xxx;          (* NaNaslovu10 *)
END Primer2.
```

Sicer pa lahko definiramo tudi celoten, recimo 16-biten pomnilnik kot polje na naslednji način:

```
CONST MaxNaslov = 177777B;
VAR   Pomnilnik [0B]:
        ARRAY [0..MaxNaslov] OF WORD;
```

4.2 Procesi in prekinitveni podprogrami

Za lažje razumevanje možnosti pisanja prekinitvenih podprogramov v moduli-2 si najprej oglejmo, kako jezik uporablja procese.

Proces je definiran kot osnovna izvajalna enota, ki se izvaja zaporedno (sekvenčno). Seveda lahko nek proces proži in ustvarja druge procese. V moduli-2 je proces poseben tip, ki ga oddaja modul SYSTEM. V dodatku k definicijam jezika [18] je ta tip sicer opuščen, vendar ga bomo v naših primerih zaradi preglednosti še uporabljali. Namesto tipa PROCESS se sedaj uporablja enostaven tip ADDRESS. Proces se izvaja zaporedno in neprekinjeno, dokler ne požene nekega drugega procesa. To izvede s podprogramom

```
PROCEDURE TRANSFER (VAR od, komu: PROCESS);
```

ki ga oddaja zopet modul SYSTEM. Ta modul tudi oddaja podprogram

```
PROCEDURE NEWPROCESS (p: PROC; a: ADDRESS;
                      n: CARDINAL;
                      VAR pr: PROCESS);
```

kjer je p podprogram, ki se bo izvajal kot samostojen proces. Svoje spremenljivke (sklad) bo imel v prostoru, ki se prične na mestu a in je velik n besed. Vse potrebne podatke o novem procesu shrani podprogram v parameter pr.

Naslednji korak je povsem razumljiv in sicer definiramo podprogram, ki poslužuje neko prekinitev kot nov proces. Problem je le še, ker ga je treba v pravem trenutku pognati s podprogramom TRANSFER. V ta namen modul SYSTEM oddaja še en podprogram, katerega definicijo bomo napisali sedaj za računalnike, ki imajo prekinitve vektorirane.

```
PROCEDURE IOTRANSFER (VAR komu, od: PROCESS;
                      vektor: CARDINAL);
```

V bistvu sistem ob kliou tega podprograma poveže proces od z vektorjem vektor in požene proces komu. Ta se izvaja, dokler se ne zgodi prekinitev in takrat se zopet požene proces od. Tako dobimo skelet prekinitvenega podprograma:

```
PROCEDURE Skelet;
BEGIN
  Inicializacija;
  LOOP
    IOTRANSFER (program, posluzi, vector);
    PosluziPrekinitev
  END
END Skelet;
```

Primer takšnega podprograma bomo navedli v naših primerih, ki bodo navedeni v drugem delu članka.

5. Zaključek

Na tem mestu zaključujemo prvi del našega prispevka. V njem smo želeli podati osnovne zmožnosti programskega jezika modula-2. V skopih besedah smo opisali osnovne podatkovne, ukazne in programske strukture jezika.

V nadaljevanju smo opisali osnovna načela modernega načina programiranja, za katerega se je v svetu udomačil izraz modularno programiranje. Zadnje poglavje prvega dela opisuje možnosti nizkonivojskega in sistemskega programiranja v moduli-2.

Zahvala

Za vso pomoč pri snovanju besedila in dragocene nasvete pri iskanju primernih slovenskih izrazov za angleške besede se najlepše zahvaljujem prof.dr.Boštjanu Vilfanu.

Literatura

[1] E.Biagioni, K.Hinrich, G.Heiser, C.Muller: A Portable Operating System Interface and Utility Library, IEEE Software, November 1986.

[2] R.P.Cook: Modula-2 Experiments Will Help Future Language Design, IEEE Software, November 1986.

[3] R.Gleaves: Modula-2 for Pascal Programmers, Springer-Verlag, 1984.

[4] J.Gutknecht: Separate Compilation in Modula-2: An Approach to Efficient Symbol Files, IEEE Software, November 1986.

[5] P.B.Hansen: The programming language Concurrent Pascal, IEEE Trans. on Software Engineering SE-1, 2 (junij 1975), 199-207.

[6] K.Jensen, N.Wirth: PASCAL User Manual and Report, Springer-Vgrlag, 1975.

[7] Modula-2 Development System, Modula Corporation Provo, Utah< 1984.

[8] Modula-2 System for Z80 CP/M, Hochstrasser Computing AG, Zurich, Switzerlind, 1984.

[9] B.Mohar, E.Zakrajšek: Uvod v programiranje, DMFA, Ljubljana, 1982.

[10] OMSI Pascal-1, version 1.2 Manual for RT-15, Oregon Software, Oregon, 1981.

[11] G.Pomberger: Software Engineering and Modula-2, Prentice-Hall, 1984.

[12] Programski jezik Fortran, Iskra Delta, Ljubljana, 1982.

[13] I.C.Pyle: The Ada Programming Language, Prentice-Hall, 1981.

[14] SDS-XP, Extended Performance Modula-2 Software Development System, Ynterface Technologies, Houston, Texas, 1985.

[15] TURBO PASCAL, Version 3.0, Reference Manual (1985), Borland International Inc., 4585 Scotts Valley Drive, Scotts Valley, CA 95066.

[16] VAX-11 Pascal Language Reference Manual, Digital Equipment Corporation, Maynard, Massachusets.

Ali bo fizika izrinila matematiko s prvega mesta med znanostmi?

Razgovor Tony Durhamsa z dr. Davidom Deutschem o možnosti, da matematiko preženemo z njenega platoničnega oblaka in da fizika zavzame njeno mesto prve med znanostmi.

Matematika je kot prva med znanostmi izzvana. Če bo prevrat uspel, tvega izgubo statusa tudi računalniška znanost, ki ta staus uživa kot veja matematike, čeprav je pri tem tveganje majhno. Izid je lahko povsem drugačen, če bi se pokazalo, da računalniśk znanost (kot običajno imenujemo znanost o računanju) sploh ni del matematike.

Izzivalec je fizika. In če fizika postane nova kraljica znanosti, bi lahko, kar je presenetljivo, računalniško znanost priznali za prvega ministra. Poseben namišljen računalnik namreč igra pomembno vlogo v miselnih poskusih, s katerimi poskušajo fiziki utemeljiti svoj naskok.

Kdo je komu potrebnejši? Fizika se je zaupala matematiki kot jeziku, v katerem je izrazila svoje najglobije in najsplošnejše trditve. In vendar so matematični zakoni v nekem smislu zelo šibki. S splošnimi izrazi opisujejo, kako se obnašajo veliki fizikalni sistemi, ne rešujejo pa problemov podrobnega simuliranja poljubnega fizikalnega sistema, ki je v bistvu problem računanja.

Na drugi strani se je matematika vedno videla živeco v platonskem svetu, na nek način presegajoco manjvredne posebnosti in izjemnosti fizikalnega sveta. Toda ta pogled je v nasprotju z materialistično komponento v znanosti, ki kot se zdi, pušča vse manj prostora za koncepte čistega, od materije neodvisnega umevanja. Za materialiste je tudi matematika le rezultat dela možganov, računalnika ali kakega drugega fizičnega sistema. Potemtakem fizika postavlja meje dosežkom matematike.

Lahko bi (in verjetno tudi bodo) napisali obsežne knjige, ki bi razlagale argumente ene in druga strani. Nobena stran pa si ne more privoščiti ošabnosti, saj je 20. stoletje pokazalo, da obstajajo stvari oziroma pojavi, ki jih ne more poznati niti matematika niti fizika.

Teorija kvantne fizike dokazuje, da načeloma ne moremo poljubno natančno izmeriti vrednosti vseh spremenljivk nekega fizičnega sistema. Če izmeriš natančno vrednost ene spremenljivke, se ti bo vrednost druge, po Heisenbergovem principu nedoločenosti, neizbežno izmuznila.

Goedlov izrek in številni sorodni rezultati v matematiki dokazujejo, da obstajajo trditve, ki so resnične, ne moremo pa jih dokazati. Nekateri domnevajo, da med obema rezultatoma obstaja globoka povezava, ki postavlja meje tako matematičnemu kot fizikalnemu znanju.

Kje ob vsem tem stoji računalniška znanost? Alan Turing je prvi poskušal določiti, kaj je mogoče izračunati s strojem. V ta namen je moral najprej ustrezno definirati stroj. Njegova definicija v določenem smislu velja za vse poznane računalnike. Logik Alonzo Church je razmišljal v podobni smeri. Oba pa sta svoje rezultate oblikovala znotraj okvirov, ki so v bistvu matematični in, v nasprotju s splošnim prepričanjem, jima ni uspelo dokazati, da meje, ki sta jih upoštevala, res veljajo za vsak računalnik. Domneva v obliki ti. Turingove-Churchove hipoteze je še vedno le domneva.

Ker je omenjena hipoteza oblikovana matematično (se pravi s sredstvi matematike), upravičeno pričakujemo, da bo tak tudi izziv oziroma ugovor proti njej. Čeprav je presenetljivo, se dozdeva, da to hipoteto lahko ovrže fizika. Pri tem velja še vedno neizrečena pa vendar upoštevna domneva, da nas matematična definicija računalnika osvobaja nepotrebnih fizikalnih omejitev. A nekateri pomembnejši zagovorniki fizikalnega pogleda menijo, da je matematika tista, ki trpi zaradi nepotrebnih omejitev. Domnevajo, da imajo fizikalni sistemi morda sposobnost računanja, ki presegajo tiste, ki jih lahko dosežemo z golo matematično manipulacijo simbolov.

Dr. David Deutsch iz Oxfordske univerze je eden vodilnih med zagovorniki pogleda, v katerem ima fizika prvo mesto. Ukvarja se s teoretičnimi raziskavami računalnikov, ki se pokoravajo zakonom kvantne fizike.

Med vsemi teorijami, ki razlagajo fizični svet, ponuja najboljšo razlago kvantna teorija skupaj z relativnostno teorijo Alberta Einsteina. Za večino ljudi so pojavi, ki jih napoveduje, nenavadni in v nasprotju z intuicijo, vendar so napovedi potrjene s poskusi. Kvantni učinki običajno ostajajo prikriti, razen kadar je sistem, v katerem jih opazujemo, zelo mrzel ali zelo majhen. Zato se zdi, da v običajno velikih toplih sistemi, veljajo zakoni klasične fizike.

Eden najbolj nenavadnih vidikov kvantne fizike je, da na potek dogajanja poleg dogodkov, ki se dejansko zgodijo, vplivajo tudi dogodki, ki bi se lahko zgodili (pa se niso). Elektroni, usmerjeni eden za drugim proti oviri z dvema režama, oblikujejo na zaslonu za oviro vzorec, ki je drugačen, kot če bi bila reža le ena (oziroma če bi izmenično zapirali po eno režo). Če lahko gre elektron samo skozi eno režo, kako že sama prisotnost druge reže, vpliva na njegovo gibanje?

Deutsch je svoj čas menil, da nalaga kvantna mehanika nove omejitve zmogljivostim Turingovega stroja. K sreči pa je ugotovil, da se moti. Stroj, ki bi ustrezal Turingovemu stroju, vendar bi bil zgrajen iz abstraktnih kvantno-mehanskih komponent, bi imel zmogljivosti, ki jih navaden Turingov nima.

Tak stroj ne more izračunati funkcij, ki niso izračunljive s Turingovim strojem, niti ni pokazano, da kakšno funkcijo izračuna hitreje. Kadar na vprašanja obstaja le en odgovor, ga vsaj kolikor nam je do danes znanega, kvantni računalnik ne bo našel hitreje kot kak drug.

Vznemirljiva razlika pa nastopi kadar rešujemo probleme. Problem ima namreč lahko več odgovorov, od katerih vsak ustreza eni od

rešitev. Deutsch je dokazal, da lahko kvantni računalnik pospeši reševanje problemov. Probleme, ki imajo pri reševanju s Turingovim strojem eksponentno časovno zahtevnost (tj. čas potreben za izračun raste eksponentno z naraščanjem velikosti problema), je s kvantnim računalnikom mogoče rešiti v času, ki je polinomsko odvisen od velikosti problema.

Na žalost je Deutsch dokazal svoje rezultate le za skonstruirane probleme, ki očitno nimajo praktične uporabe. Kljub temu imajo njegova dognanja vsaj simboličen pomen; k uporabnejšim rezultatom nas bodo pripeljala pozneje.

Kvantni računalnik sam seveda obstaja le teoretično. Deutsch trdi, da je njegov model v skladu s fizikalno teorijo, čeprav je abstrakten in ne določa, kakšni so v njem sodelujoči sistemi: so to molekule, superprevodne zanke, ali elektroni ujeti v "kvantne vodnjake"? Tehnologija bo morda tudi temu problemu nekoč kos.

Morda najbolj nenavaden pri tem teoretičnem računalniku je način, s katerim dosega hitrost. Deutsch ga imenuje "kvantni paralelizem". Povedano enostavno: stroj obstaja hkrati v več vzporednih svetovih. V vsakem od njih izvaja rahlo drugačno računanje. Na koncu se rezultati združijo in uskladijo.

Deutsch verjame "Everettovi interpretaciji" kvantne mehanike, ki so jo popularizirale knjige, kot je npr. Ostali svetovi (Paul Davies, Other Worlds). Za običajne interpretacije je obstoj vzporednih svetov le možnost ali celo izmislek fizikov, ki naj pomaga pri določenih izračunih. Everettova interpretacija pa pravi, da so vzporedni svetovi resničnost.

Sprva so menili, da je razlika v interpretaciji bolj filozofske kot fizikalne narave. "Veljalo je mnenje, da gre le za različne interpretacije istega formalizma, saj so vse interpretacije predvidevale enake eksperimentalne rezultate", pravi Deutsch, "toda menim, da je bila Everettova interpretacija celo tedaj neizogibna, saj so po mojem ostale interpretacije pomanjkljive s filozofskega stališča. Seveda pa to ne more biti argument za fizika."

"Toda v zadnjih letih" nadaljuje Deutsch, "se je pokazalo, da je ime interpretacija napačno. Everettova verzija kvantne teorije se rahlo razlikuje od običajne, njeno različnost pa lahko, vsaj v načelu, pokažemo s poskusi. Domnevam, da sem prvi, ki je objavil možnost takega poskusa".

Deutsch smatra, da njegov miselni poskus s kvantnimi računalniki zahteva odločnejši odgovor, kot znani poskus z dvema režama. "Pri računanju nastane nekaj novega", pravi. "V vsaki veji nastane novo znanje, nova, koristna informacija. In težje kot v primeru poskusa z elektronom filozofsko rečemo, da gre za različne možnosti, ki so kombinirane ena z drugo, dale končni rezultat. V določenem smislu gre res le za različne stopnje, v drugem pa se teže izmikamo, saj nastane nova informacija. Od kod je prišla? Kdo jo je oblikoval?"

Zares težko bi zanikali obstoj vzporednega sveta, ki bi za nas opravljal koristno računanje.

Obstaja domneva, da vsebuje množica problemov, ki jih obvladamo s kvantnim računalnikom (Deutsch jih je imenoval qp-problemi), kot svojo podmnožico eno od posebnih skupin problemov, neobvladljivih s Turingovim strojem.

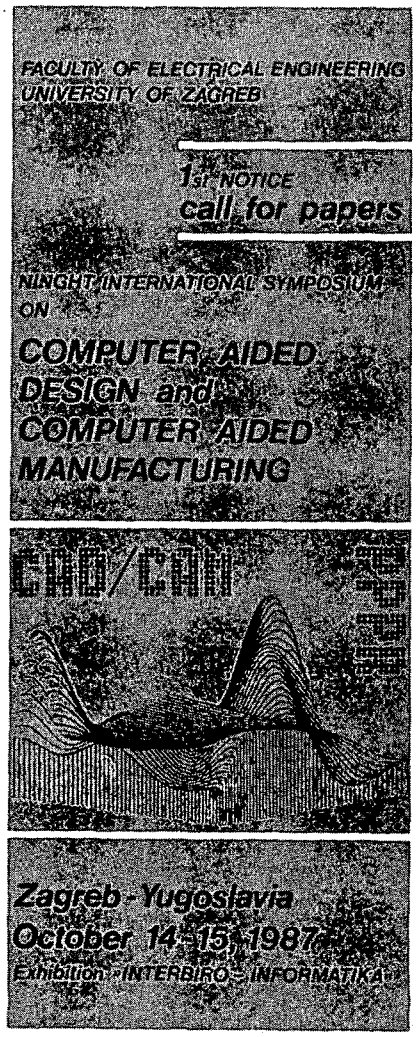
To so np-kompleti oziroma npc-problemi. Pomembni so zato, ker bi hiter algoritem za reševanje kateregakoli npc-problema avtomatično zagotovil rešitev vrste drugih, domnevno nerešljivih problemov.

"To je", kot pravi Deutsch, "zelo vznemirljiva možnost, ki je ne smemo izključiti. Ne vemo. Toda odkritje vsaj enega qp-problema, ki bi bil hkrati tudi npc-problem, bi popolnoma spremenilo računalniško znanost in nas celoten pogled na svet, saj bi pomenilo, da obstaja način za reševanje doslej nerešljivih problemov."

A žalostno dodaja: "Imam zanesljiv občutek, da stvar ne bo delovala".

Naj bo kakor koli. Delo Deutscha je že prispevalo svojo težo ideji, naj računalniška znanost sloni bolj na fiziki kot na matematiki. In danes je računalniška znanost pomemben del intelektualne sfere. Matematiki bodo morali zbrati svoje intelektualne sile, če želijo ostati na vrhu drevesa znanosti.

Computing, December 11, 1986
Prevedel Ivo Ščavničar



CAD has become the initial module of Computer Integrated Manufacturing – CIM, wide spread in a great number of industries.

The Symposium CAD/CAM 1987 *once again provide* all engineers, doctors, researchers and scientists interested in theoretical and applied work on CAD and CAM or whose work involves application of computers in research and education, with good opportunity to exchange experience and knowledge *through talks, discussions and* meetings.

SUBJECT OF THE SYMPOSIUM

A. CAD/CAM system components

A1. System software, computer architecture
A2. Automatic, robotic, flexible manufacturing systems
A3. Analysis, synthesis and optimization techniques
A4. *Information structures,* business systems

B. CAD/CAM application areas

B1. *Architecture and civil engineering*
B2 Shipbuilding and mechanical engineering
B3 Ecology, meteorology, oceanology
B4. Electric-power and electro-mechanical engineering
B5. *Electronics*
B6. Medicine, medical information systems
B7. Forestry, wood industry

SUBMISSION OF PAPERS AND DEADLINES

Registration forms, duly filled in, with summaries up to 500 words, which illustrate the content and purpose of the paper, should be submitted by **February 16, 1987,** to the address of technical organizer – ATLAS, Congress Department. *The selected papers, written in accordance* with the instructions (which the authors will receive with notification on the provisional acceptance of the paper on the basis of the summary), should be sent in by **April 24, 1987.** Paper longer than six pages will not be accepted. The speakers will have at their disposal a slide-projector, overhead-projector, video recorder and video projector (VHS system).

REGISTRATION FEE

Registration fee amounts to US doll 100, and covers participation to the Symposium and a copy of the Proceedings. Payment should be made by a bank cheque addressed to ATLAS, and sent to the address of technical organizer.

LOCATION

The Symposium will be held in the Congress halls of the Zagreb International Fair, Boris Kidrič Avenue No 2, Zagreb.

EXHIBITION

The Symposium will be part of a specialized event INTERBIRO – INFORMATIKA, to be organized by the Zagreb International Fair from October 12–16, 1987. All participants will have free entry to the exhibition, where manufacturers from Yugoslavia and abroad will be exhibiting the latest achievements in the field of computer technology and CAD/CAM equipment.

LANGUAGES

The official languages at the Symposium are English and Yugoslav languages with simultaneous interpretation.

HOTEL ACCOMMODATION

Accommodation for participants will be provided in the hotels of A and B category. Participants wishing accommodation *should send in their hotel reservation forms* by **April 30, 1987,** following which price lists for accommodation will be received and booking made.
Please note that the number of single rooms is very limited, therefore your indication with whom you would like to share a *double room will be appreciated.*

PRESIDENTSHIP OF THE ORGANIZING AND PROGRAMMING COMMITTEE

President

Dr Zijad Haznadar, Professor at the Faculty of Electrical Engineering, University of Zagreb

Vice-President

Dr Vesna Jurčec, Adviser at the Hydrometeorological Institute, Zagreb

Secretaries

Mr Sead Berberović, Assistant Lecturer, Faculty of Electrical Engineering, University of Zagreb
Dubravka Pavlović, ATLAS, Travel Agency, Congress Department Zagreb

ADDRESS OF THE TECHNICAL ORGANIZER

ATLAS – Congress Department
Trg senjskih uskoka 7
41020 Zagreb, Yugoslavia
Phone: 41/525-333, 528-094
Telex: 22413 atlcon yu
Telefax: 41/525-468

MIPRO 87

SYMPOSIA on New Generations of Computers - NG

Sponsor: SOZD ISKRA, DO "ISKRA DELTA" LJUBLJANA

Rijeka, 18-22 May 1987
OPATIJA, "Adriatic" Congress Centre

Topics: New architecture - experiments and results of measurement, characterictics evaluation, parallel work problem, parallel systems programming, programming languages, man-machine communication, artificial intelligence, scientific applications, and other applications where high computer processing power is needed.

Co-ordinators:
Prof. dr. Anton Zeleznikar, mr. Petar Brajak, DO "ISKRA DELTA", 61 000 LJUBLJANA, Parmova 43, tel. (061) 312-988

Thursday, 21. 05. 1987

08.00-14.00 **NG-YU-MAGIC (invited papers)**

Prof. dr. Suad Alagić, ETF Sarajevo: CONCEPTUAL MODELLING-OBJECT ORIENTED APPROACH;

Prof. dr. Jozo Dujmović, ETF Beograd: COMPARISON AND PERFORMANCE OF HIGH-LEVEL LANGUAGES FOR NEW GENERATION OF COMPUTERS

Prof. dr. Veljko Milutinović,Purdue University, West Lafaette, Indiana, USA: MICROPROCESSOR DESIGN FOR GaAs TECHNOLOGY;

Prof. dr. Branko Souček, University of Arizona, Tuscon, Arizona, USA: 6-th GENERATION COMPUTERS;

Prof. dr. Dalibor Vrsalović,Carnegie-Mellon University, Pittsburgh, USA: PARALLEL ARCHITECTURES AND PERFORMANCE ANALYSIS;

Prof. dr. Anton P. Zeleznikar, ISKRA DELTA Ljubljana: ARTIFICIAL INTELLIGENCE EXPERIENCES ITS OWN BLINDNESS.

14.00-16.00 Lunch

16.00-17.30 Papers - group "architecture"

A. P. Zeleznikar: STRATEGY OF COMPUTER INFORMATION IN THE 1990's
S. Prešern: A SURVEY OF PARALLEL COMPUTER ARCHITECTURES
P. Brajak: PERFORMANCE ANALYSIS OF THE INTELLIGENT MEMORY MODULE BASED PARALLEL MACHINE
J. Novak: PROCESS SCHEDULING AND SINHRONIZATION USING FETCH & ADD INDIVIZIBLE OPERATION
L. Vogl: VLSI DESIGN COMPLEXITY OF THE INTELLIGENT MEMORY MODULE
S. Jeram: COMPUTATION MAY HINGEON THE BIOLOGICAL MATERIAL

17.30-17.45 Pause

17.45-19.45 NG-Round table

MIPRO 87

Friday, 22. 05. 1987

09.00-10.15 Papers - group "architecture"

J. Silc, B. Robic: PARALELNO PODATKOVNO VODENJE RACUNALNISKE ARHITEKTURE
R. Jankovic: PROCENA PERFORMANSE CROSSBAR-MATRICE SA PROGRAMSKIM UPRAVLJANJEM
P. Kolbezen, B. Mihovilovic: RISC ARHITEKTURA I NJENA APLIKACIJA
P. Kolbezen, S. Mavric: PARALELNO VEKTORSKO PROCESIRANJE
S. Mavric, A. Brodnik, R, Trobec, M. Spegel, P. Kolbezen: PS-11 VECPROCESORSKI SISTEM NA VODILU Q

10.15-10-30 Pause

10.30-11.15 Papers - group "software tools & performance analysis"

M. M. Miletic: PROLOG PERFORMANCE ON DEC ARCHITECTURES
B. Mihovilovic, P. Kolbezen, I. Silc, B. Robic: MODELIRANJE VECPROCESORSKEGA SISTEMA S POMOCJO GRAFNEGA MODELA RACUNANJA
Z. Dolenc: JEDAN PRISTUP FPM METODI U ANALIZI OPCE ZATVORENE MREZE REPOVA

11.15-11.30 Pause

11.30-12.30 Papers - group "applications"

S. Presern, R. Murn, D. Pecek: ARCHITECTURE FOR COMPUTER-VISION .
Z. Delic: POUZDANOST KOMUNIKACIJE PROGRAMER RACUNALO
Z. Sijercic, M. Partalo: MOGUCNOSTI IZVEDBE ALGORITMA DINAMICKE TRANSFORMACIJE VREMENA JEDNOG PARALELNOG SISTEMA U OBRADI SLIKE
M. Novakovic: MOGUCNOSTI PRIMENE JEDNOG PARALELNOG SISTEMA U OBRADI SLIKE

For more information: Mr. Jadranko Novak Dseit - Mipro, Trg P. Togliatti 4/I, 51 000 Rijeka, tel.: 051/30-491

AVTORJEM CASOPISA INFORMATICA

Zaradi možnega plasmaja časopisa Informatica v tujini naprošamo avtorje časopisa Informatica, da pišejo v prihodnje svoje članke, referate, eseje in novice po možnosti v angleščini. Povzetek v domačem jeziku z naslovom prispevka naj bo dodan na posebnem listu v formatu stolpca (47 znakov na vrstico).

AUTORIMA CASOPISA INFORMATICA

Zbog mogućeg plasmana časopisa Informatica u inostanstvu, autori časopisa Informatica se umoljavaju da pišu u buduće svoje članke, referate, eseje i novosti po mogućnosti na engleskom. Abstrakt u domaćem jeziku sa naslovom članka neka bude dodat na posebnom listu i izpisan u formatu stupca (47 znakova na redak).

TO AUTHORS OF INFORMATICA

To achieve the possible exchange of Informatica with foreign journals, the authors of articles for Informatica are kindly requested to submit their articles, papers, essays, and news in English, when possible. Abstracts in a domestic language containing the title of an article should be submitted on a separate sheet and printed in the obvious column format (47 characters per line).

Tiskarna Kresija