ISSN 0136-3549
0320-3409

TALLINNA
POLÜTEHNILISE INSTITUUDI
TOIMETISED

614

ТРУДЫ ТАЛЛИНСКОГО
ПОЛИТЕХНИЧЕСКОГО
ИНСТИТУТА

TPI '86

DATA PROCESSING,
COMPILER WRITING,
PROGRAMMING

614

TPI '86

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED

ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

UDK 681.3.06.

DATA PROCESSING, COMPILER WRITING, PROGRAMMING

Transactions of the Faculty of Economics
LIX

Tallinn 1986

Contents

No 614

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED
ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

B. Tamm, E. Tyugu
UDK 681.3.06.65.015.11

KNOWLEDGE APPLICATION IN CAD AND CONTROL

Abstract

Knowledge representation and processing is presently a major concern in the studies of artificial intelligence. In this paper we attempt to discuss some elements of knowledge processing applied to CAD and control.

1. Introduction

Knowledge representation and processing is presently a major concern in the studies of artificial intelligence. A number of companies have been established for software production to develop programs, using knowledge bases for problem solving. Expert systems used for geological exploration [1], diagnosing in medicine [2] and in engineering [3], in system design [4] are more impressive.

We attempt to discuss some elements of knowledge processing applied to CAD and control, showing some particular examples. The corresponding systems enjoy several merits:

- adaptivity to the environment changes
- provision of computer interaction convenient to the user, in particular, communication in some language close to the natural one and a dialogue on an adaptive scenario
- ease of extension by addition of new knowledge
- high reliability due to common nucleus which remains unaltered with the knowledge changing.

2. What is knowledge?

The concept of machine knowledge was introduced into the field of artificial intelligence in the mid 1960s, when question-answer systems [5,6] and task solvers [7,8] were developed in a number of dissertations. Knowledge was represented in the form of certain networks, particularly, through semantic networks. In the mid 1970s a further major step was made when M. Minski introduced the concept of a frame, implying knowledge module of an object or a situation. A review on knowledge representation is given in the paper [9] . However, this concept is not adequately defined, and the question of "What is knowledge?" - might be answered in one of the following ways:

- text (i.e. data) in the knowledge representation language
- data controlling computations in different kinds of programs
- programs developed automatically at problem solving without the human task supplier
- object and situation description
- information to be used at the purposeful action in different situations.

The first definition directly implies the semantics of the knowledge representation language which is difficult to be provided. The second and third definitions are too narrow with respect to orientation to programs and computations. For the fourth definition the description concept is to be further specified. The fifth definition associates knowledge with information usefulness. Thus, not a single definition considered is adequate. Let us recall how difficult it is to give a satisfactory definition for "data", although everyone involved in computing knows from his own experience what data is. The knowledge content is to be derived from one's experience, for instance, from a discussion of examples. Prior to the discussion of the examples of knowledge application in various fields, we will describe their basic properties, briefly featured in the answers above.

1. Knowledge can be represented in the data form, in particular, in the text form in a formal language, in the network form, providing different kinds of connections between knowledge elements. There exists the problem of knowledge translation from one representation form to another. It is a matter of technology, since each knowledge representation language claims to be universal in a sense. At least they assure extension by adding new relation and program types, "elementary meanings" or new connection types. Language expressiveness gives ground to the convenience of language application, since knowledge representation languages worked out in detail, are always developed for a particular application field.

2. Knowledge has the capability of controlling information processes (computations). It means that in the knowledge application system process flow is determined by knowledge and is almost independent of system structure. The latter determines only the constraints for process run (with regard to qualitative characteristics). Here lies the essential difference between knowledge and, for instance, numerical data, implemented to control a dynamic object.

3. Knowledge may encompass a procedural component, i.e. programs. But the application of these programs is controlled by knowledge. For instance, parameter linkage and program start is performed automatically within a knowledge application system without the human who started the process.

4. Knowledge is divided into separate fragments: descriptions of objects, processes, situations, events. These fragments (knowledge modules) are called frames. Methods for running frames have been developed. They are similar to a great extent to the methods employed for classes and for describing abstract data types in programming languages. But they differ from abstract data types, since frames always have a close connection by means of mutual references.

3. Knowledge in Control Systems

Let us consider a model of a heat exchanger shown in Figure 1. The parameters include bulk velocities V_1, V_2, temperature differences DT_1, DT_2, specific heat capacities c_1, c_2 and heat quantities Q_1, Q_2. Stationary conditions are described by the classroom equations:

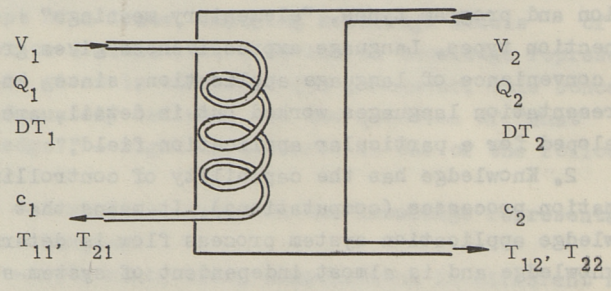


Figure 1

$$Q_1 = Q_2,\ Q_1 = V_i DT_i c_1,\quad DT = T_{2i} - T_{1i}.$$

In the knowledge representation language, for instance, in the input language of PRIZ system this model can be expressed in the text form:

```
HEAT_EXCHANGER: (V1, V2: BULK_VELOCITY;
                 T11, T12, T21, T22: TEMPERATURE;
                 Q1, Q2: HEAT;
                 DT1, DT2: TEMPERATURE_DIFF;
                 c1, c2: SP_HEAT_CAPACITY
                 Q1 = Q2;
                 Q1 = V1 * DT1 * c1;
                 Q2 = V2 * DT2 * c2;
                 DT1= T21 - T11;
                 DT2= T22 - T12);
```

In the simplest case, the concepts of bulk velocity, temperature, heat quantity, temperature difference and

specific heat capacity can be specified as real values:

```
BULK_VELOCITY, TEMPERATURE, HEAT, TEMPERATURE_DIFF,
SP_HEAT_CAPACITY:REAL;
```

Any one concept can be described in more detail. For instance, we could have described temperature difference by the following frame:

```
TEMPERATURE_DIFF:(DT:REAL;
                  VIRTUAL T1,T2: REAL;
                  DT = T2 - T1),
```

and then omit in the description of the heat exchanger variables T_{11}, T_{12}, T_{21}, T_{22} and the last two equations, since they are involved in DT_1 and DT_2.

The same knowledge can be provided, describing the heat carrier concept:

```
HEAT_CARRIER:(Q: HEAT;
              DT: TEMPERATURE_DIFF;
              V: BULK_VELOCITY;
              c: SP_HEAT_CAPACITY;
              Q = V * DT * c);
```

and through that concept a heat exchanger can be specified:

```
HEAT_EXCHANGER:(A1,A2: HEAT_CARRIER;
                A1 * Q = A2 * Q.)
```

The amount of knowledge provided in the last three descriptions is equal to that of the first one, only now the descriptions are shortened, and instead of one, three concepts are specified: temperature difference, heat carrier and heat exchanger.

Thus, knowledge representation languages enable us to provide mathematical models in an adequately natural form. Actually, the first heat exchanger description can be considered as: 1) a record of a mathematical model containing also explanations of the values V_1, V_2, T_{11}, etc.; 2) frame (knowledge module) of a heat exchanger; 3) an abstract data type or a class description, as it is referred to in programming.

Frame representation of knowledge on an object is convenient also because input-output variable models are not fixed. Any one of the twelve variables in the heat exchanger model can be both an input or output one, depending on the task being solved. Using this background, over 16 million tasks with different sets of input-output variables can be formulated. Certainly there are considerably fewer meaningful tasks and still fewer solvable ones. The universality of knowledge representation in the form of computation frames is proved right here. If the heat exchanger frame is used concurrently with other knowledge for solving more complex tasks, then any task may come up in the course of decision retrieval, since the problem of meaningfulness of a task in a computer is more difficult to be judged than its solvability is to be checked.

We will proceed with the problem of process description in control systems. The processes going on in the modules can be given in the module descriptions. Let us discuss the process frame involved in the whole system.

We shall use general equations of a dynamic system:

$$X' = \Phi(X,U,t)$$

$$V = \Psi(X,U,t)$$

where X - system state in the moment t,

X' - system state in the next observed moment,

U,V - system input-output values in the moment t.

Let us proceed to the recursive relations for $i = 0,1, \ldots$

$$X_{i+1} = \Phi'(X_i,U_i,t_i)$$

$$V_i = \Psi'(X_i,U_i,t_i),$$

$$t_{i+1} = t_i + \Delta t.$$

We shall include process start, time variation, transition to the next process state in the process concept. Functions Φ' and Ψ' will be described by the system frame, presented in the process concept in the

form of an indefinite type component MODEL. Let us use the following notations:

TØ - initial time moment

T2 - end time moment

TM - step in time

XØ - initial state of the system

X - current state of the system

X1 - next state of the system

```
PROCESS:(MODEL:INDEF';TØ, T2, TM: REAL
                XØ, X, X1: MEMORY'
               RESULT: MEMORY'
```

*The component RESULT involves the form of process representation (table, diagram, etc.).

```
MODULE' COMPOSITION INP'MODEL.ALL'. INITSTATE
                    OUTP'X=     XØ
MODULE' COMPOSITION INP'MODEL.ALL'. NEXTSTATE
                    OUTP'X1;
MODULE' DECOMPOSITION INP'X
                    OUTP'MODEL.ALL'STATE;
MODULE' MODELLING INP'TØ,T2,TM,XØ
                    OUTP' RESULT
                    ARG' X
                    RES' X1).
```

The first two relations in the process concept pick up the initial state and new state of the system from the states of system elements. The third relation decomposes system states into states of components. It is assumed that all subsystems and system modules which are its components have standard names for the initial state (INITSTATE), next state (NEXTSTATE) and current state (STATE).

The process concept described in this way suits for modelling the behaviour of a constant step in time system. If the program MODELLING only attributes the value X1 to the variable X for the next time moment, then the program is appropriate for the modelling systems with elements of

independent behaviour. But the program MODELLING can be written also in a more specific way. For instance, for the systems in which different parts consume common resources, and thus the transition to the next state of each separate element depends on the states of all system elements. These are specifically ecological and demographic systems.

4. Knowledge in CAD

Though a great variety of CAD systems are available, they are all oriented on knowledge. CAD developers are avoiding the term "knowledge" and are using knowledge processing methods spontaneously, but researchers of artificial intelligence have accepted CAD as an area for approbating their methods.

An analysis of the performance of a design engineer shows the following.

1. A multitude of relatively small tasks are solved in the design, each of which is readily formalized and solved on a computer. The difficulty lies in the diversity of these tasks, specifically, in the fact that it is inconceivable to write ready made "strict" programs to solve any feasible tasks.

2. The design task discussed in its entirety, does not subject to algorithmization, but there exist engineering design methods based on knowledge application (undergraduates are taught these methods on the examples of practical projects).

3. Engineering graphics is of considerable importance in the design. In CAD computer graphics and geometric modelling correspond to it. The latter in conjunction with recognition and visualization of complex spatial forms is an application area for the methods of artificial intelligence.

4. The bulk of engineering tasks are ambiguously described and therefore expert knowledge is required. A common engineering calculation involves multiplication of the calculation result by the correction coefficient (which can vary from 0.5 to 10 or more), but coefficient selection is based on expert knowledge.

5. Engineering objects are, as a rule, well structured. With the provision of their elements' and connections' description (i.e. relations between objects), models of any complex design objects can be built.

Knowledge processing systems related to simple data rather than simple data themselves, where for the first time procedure sequences were described in the program text and synthetized, were intended for calculating cutting tool movement and technological data. Thus, to solve a task described in a certain command, both subsequent and previous commands had to be considered simultaneously in the task description. The next example is related to the SAP-2 [10] (Fig. 2) and illustrates the determination of a geometric point on the basis of data and rules of its description, using data and rules of other geometric elements which have indirect relation to the unknown point.

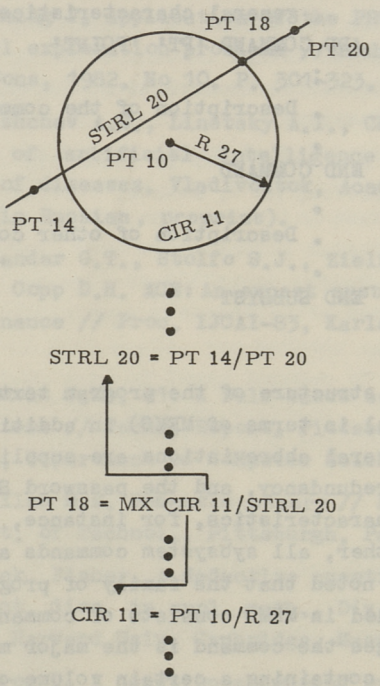


Figure 2

Presently CAD systems are built up on complex programming tools [11,12], comprising adequately developed means for knowledge processing. Let us discuss briefly the metamonitoring system MEMO [13]. MEMO includes the MEMOGEN language to describe new subsystems and extend the existing ones. Let us assume that we have to add a new subsystem (SS) which enables to make the required geometric calculations, into an application package or into an application system at our disposal. The new system called "GEOMETRY" can readily be described in the MEMOGEN language, as follows.

```
* MEMOGEN
    ADD SUBSYST 'GEO' 'GEOMETRY' PASSWORD 'G84'
      SEPARATORS GLOB ',' SPEC ';'
        .
        . Descriptions of other
        . general characteristics of the SS
      ADD COMMAND 'PT' 'POINT'
        ;
        . Description of the command 'PT'
        .
      END COMMAND
        .
        . Description of other commands of the SS
        .
      END SUBSYST
* END
```

In the structure of the program text (called a language model in terms of MEMO) in addition to the names SS one or several abbreviations are supplied which reduce information redundancy, and the password SS. A description of general characteristics, for instance, of parameters follows. Further, all sybsystem commands are described. It should be noted that the family of programming languages included in MEMO consists of command languages. In these languages the command is the major module or language frame, containing a certain volume of knowledge on the object characterized by it, and an inner logical

structure which provides for the application of these knowledge in accordance with the given initial conditions. A detailed description of such command is given in [14].

5. Conclusion

The knowledge processing systems discussed above enjoy the merit of extending the class of solvable tasks and a substantial reduction of expenses on programming.

This paper covers only a few applications of knowledge processing in engineering and manufacturing, but we hope it has demonstrated that new means for representation of facts and regularities, and new programming methods are effectively applied to solve diverse design and control tasks.

References

1. Gashing J. Application of the PROSPECTOR system to geological exploration problems // Machine Intelligence. J. Wiley & Sons, 1982. No 10. P. 301-323.

2. Kleshchev A.S., Linetsky A.I., Chernyahovskaya M.Y. Application of artificial intelligence methods for diagnostics of diseases. Vladivostok, Academy of Sciences, USSR, 1978 (in Russian, preprint).

3. Vesander G.T., Stolfo S.J., Zielinski J.E., Miller F.D., Copp D.H. ACE: An expert system for telephone cable maintenance // Proc. IJCAI-83, Karlsruhe, 1983. P. 116-121.

4. Dermolt J.Mc. R1: A rule-based configurer of computer systems // Techn. Report. Pittsburgh-Carnegie-Mellou Univ., Department of Computer Science.

5. Quillan M.R. Semantic memory // Doct. diss., Carnegie Inst. of Technol., Pittsburgh, Pa., Oct. 1966.

6. Black, Fisher. A deductive questions-answering system // Doct. diss. in appl. math., Div. of Eng. and Appl. Phys., Harvard Univ. Cambridge, Mass., June 1964.

7. Bobrow D. A question-answering system for high school algebra word problems // Proc. AFIPS Ann. Fall Joint Computer Conf., 1964. P. 591-614.

8. Martin W.A. Symbolic mathematical laboratory // Doct. diss., MIT. Cambridge. Mass., Jan. 1967.

9. Kleshchev A.S. Knowledge representations. Methodology, formalism, calculations handling and programming support // Applied Informatics. Moscow, 1983. P. 49-94 (in Russian).

10. Tamm B.G. Description of the SAP-2 language for programming machine-tool operation // Algorithms and Algorithm Languages. 1970, No 5 (in Russian).

11. Tamm B.G., Kyttner R., Pruuden J. An approach to integrated CAD systems implementation // Advances in CAD/ /CAM. North Holland P.C., 1982. P. 315-334.

12. Kalja A.P., Kahro M.I., Tyugu E.H. Instrumental programming system ES EVM (PRIZ ES). Moscow, 1981 (in Russian).

13. Pruuden J., Markush A. Monitoring system SAPR for computer-aided design in manufacturing technology // Computing in Socialist Countries. No 11. Moscow, 1982. P. 57-63 (in Russian).

14. Tamm B.G. Data base in automatic control // Applied Informatics. No 1. Moscow, 1984 (in Russian).

No 614

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED
ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

L. Vyhandu
UDK 681.3.06

FAST METHODS FOR DATA ANALYSIS AND PROCESSING

Abstract

Some effective methods to open the structure of multidimensional data are described. The notion of standardized data base schema is introduced to provide automatic generation of application programs.

This paper presents some ideas of theory and practice, which have been useful for the author and his colleagues at the Department of Data Processing of Tallinn Technical University. Our concern was to build an effective data analysis and processing package, having powerful graphical data representing methods for creative analysis as well as different report generators suited to users' standard administrative needs.

To achieve both goals, our methods can be described as using the main part of inner degrees of freedom of data, to quickly reach a crude solution. This solution can be brought (transformed) to the required accuracy (quality).

The main tools to do it are

- fast orthogonal transformations
- theory of monotonic systems developed by our group
- standardized data base schemas with automatic program generation.

1. Data analysis using fast orthogonal transformations

Let us have a NxM-matrix A, representing data for N objects with M variables. With computer use we need O (NM) operations just to have a look at the unknown data. Therefore any method of analysis has to be slower than this lower bound.

If data have been measured at least on interval scale, different least squares methods can be used to open the structure of M-dimensional objects. There are such well-known methods as Principal Component Analysis, Factor Analysis, Clustering, Pattern Recognition, Multidimensional Scaling. Those techniques are nowadays standard and any data analysing package has them.

But aren't there methods to give a general view of data in much shorter time? We want to get results which can compete with PCA, FA, C and MDS. Yes, there are! They use well-known orthogonal transformations [1] with some additional handling [2].

To reach a maximum speed for M-dimensional data structure representation, first we use fast Haar transformation and then rotations of Jacoby type. Fast Haar transformation takes only O (NM) operations. It is recursively defined as follows:

$$D_1 = (I) \qquad D_2 = \begin{pmatrix} I & I \\ I & -I \end{pmatrix} \qquad D_{2k+1} = \begin{pmatrix} D_{2k} & 2^{k/2} I_{2k} \\ D_{2k} & -2^{k/2} I_{2k} \end{pmatrix}$$

where I is a unit matrix of order p.

Through Haar transformation B=AD the columns of B are much more orthogonal than those of A. To make them still more orthogonal we use additional Jacoby type rotations of columns of B. Taking two columns of B p and r such that $b_p > b_r$ (b_p and b_r are sums of squares of elements of columns p and r), we can make two columns orthogonal using rotation angle

$$\tan 2\varphi = 2 \sum b_{ip} \cdot b_{ir} / (\sum b_{ip}^2 - \sum b_{ir}^2).$$

But we do not rotate the vectors p and r automatically.

Namely, the sum of squares for vector p will grow by the amount

$$\Sigma(b_{ip}^2 - b_{ir}^2) \sin^2\varphi/(I-2\sin^2\varphi).$$

If the change is too small, we do not rotate at all. Moreover, in view of the precision of graphical representation little work is needed to obtain the first eigenvectors with adequate accuracy.

Another way is to use Thurstone's diagonal method [3] directly on B-matrix (not on correlation matrix) to represent the objects in a low-dimensional space.

We have also developed a very efficient method for multidimensional scaling without using gradient methods. Taking quadratic splines and optimizing coordinate-wise, we have achieved excellent results [4].

2. Nominal data ordination with orthogonal transformations

Let us now have a NxM data table A with nominal data. We define a frequency transformation for A as follows. For every variable we take its histogram and change every value $a_{ij} = h$ to its frequency f_{hj} in the histogram. The row sums describe the conformity of objects in the data system.

The new matrix Z is called the frequency matrix of a data matrix. If the number of categories for variables differs, we have to multiply frequencies by the number of categories $a_{ij} \rightarrow l_j z_{ij}$. We get an equalization of frequencies for all columns of Z. (In practice we keep the original data naturally unchanged and use histograms of all variables directly in computations).

Using either Hadamard or Haar transformation and the strategy of section 1, we get one-, two- or more-dimensional ordination for nominal data. The importance of every coordinate is measured analogously to the importance of principal components.

Another way to get interesting results is to use the scale of influence.

We define a measure of variation for every object as a sum

$$S_i = \sum_{j=1}^{m} z_{ij}^2.$$

The larger the sum S_i the more conform to group behavior is object i. Further we will define a measure of variation for the whole system as

$$S = \sum_{i=1}^{N} S_i.$$

We define the influence of the object i on the set of objects as a change in the sum of squares S_1 when object i is eliminated from the system. It is easy to find that the influence of object i can be calculated as

$$\pi_i = \sum_{j=1}^{M} \pi_{ij}, \text{ where}$$

$$\pi_{ij} = 2z_{ij}^2 - 3z_{ij} + 1.$$

The set of numbers $\pi(i)$, i=1, ..., N is called a scale of influence.

It is easy to see that taking a series of transformations

$$A \rightarrow Z \rightarrow \pi \rightarrow \pi H$$

we can use orthogonal transformations to open the structure of multidimensional nominal data.

3. Monotonic systems in data clustering

Classical clustering methods are fairly slow and some difficulties occur in interpretation of clustering results. For the last twelve years our team has successfully used the monotonic systems theory for multidimensional data structuring. Here are some general ideas of this method.

Let us suppose that there is a system W with a finite number of elements. Each element has a numerical

measure of its weight (influence) in the system. Further let us suppose that for every element $\alpha \in W$ there is a feasible discrete operation which changes as well as the weight of α and the weights of any other element β of the system. If the elements in W are independent, then it is natural to suppose that a change in the weight of α does not change the value of another element β.

System W is called monotonic, if the operation of weight change of any element $\alpha \in W$ brings about changes in the weight levels of other elements only in the direction in which α itself is changed.

To use the method of monotonic systems we have to meet three conditions.

1. There has to be a function π which gives a measure (weight) $\pi(w)$ of influence for every element w of the monotonic system W.

2. There have to be rules f to recompute the influences of the elements of the system in case there is a change in the weight of one element.

3. The rules for influence recomputing have to be commutative.

These conditions leave a lot of freedom to the researcher to choose the influence functions and rules of influence change in the system. The only constraint we have to keep in mind is that the functions f and π have to be compatible in the sense that after eliminating all elements w of the system W the final weigths of $w \in W$ must be equal to zero.

We study all $2^{|W|}$ subsets of the set W. Let $\alpha \in H \subset W$ and $\pi^{+}H(\alpha)$ or $\pi^{-}H(\alpha)$ be the value of function π on the element α. We define a kernel $H^{\ominus}$ (or $H^{\oplus}$) of a system W as a subset of W on which there is global maximum of function F of subsets H

$$F_{-}(H) = \min_{\alpha \in H} \pi^{-}H(\alpha)$$

or global minimum of function $F_{+}(H) = \max_{\alpha \in H} \pi^{+}H(\alpha)$.

The main theorem guarantees finding of the so-called determining sequence which defines exactly the extremal subset of W.

We will demonstrate how to use this theory on data matrices.

Let us have a NxM nominal data matrix A. If we take the influence function for a data element a_{ij} as $\pi_{ij} = 2z_{ij}^2 - 3z_{ij} + 1$, then we can define different monotonic systems on our data matrix:

- objects (rows of the data matrix)
- objects and variables (rows and columns of the data matrix)
- elements of the data matrix.

Changes in the algorithm dependent on different monotonic systems are trivial.

For simplicity we describe here very briefly but without any programming shortcuts only the first case (object clustering), using plus-influence.

A1. Find the sums $P(i) = \sum_{j=1}^{M} \pi_{ij}$.

A2. Find $R = \max_i P(i)$ with index k.

A3. Copy object k as a new object into the system.

A4. Label object k as taken and calculate new influences P(i).

A5. Find $R' = \max_{i(i \neq k)} P(i)$ with index k'.

A6. If $R' \geq R$ then go to A3.

A7. All the objects from step A3 belong to the kernel.

A8. If there are more objects, eliminate the first kernel and go to A2.

Our practice has shown that for interpretation it is best to use both objects and variables as elements of the monotonic system.

If the data are real numbers, we shall use as an influence function for a data element

$$g(a_{ij}) = a_{ij} + R_i + C_j,$$

where R_i is the sum of i-th row and C_j - sum of the j-th column.

For i-th row we have an influence function $G(i) = \sum_{j=1}^{M} g(a_{ij})$ and for j-th column $G(j) = \sum_{i=1}^{N} g(a_{ij})$.

For a multiplicative case one can take as an influence function

$$g'(a_{ij}) = a_{ij}(R_i - a_{ij})(C_j - a_{ij}).$$

4. Effective data processing systems building

To use data base systems directly is not enough. The software build-up for a given client must be evolutionary. In practice a typical data base will stabilize after initial booting in 2-4 years.

To speed up the design and to shorten the tuning-in process, we have developed special technologies. They are called principles of "lazy programming" and "view of the innocent bystander".

Using the first principle we practically never solve a problem in the way our client sees it. We generalize it into some class of tasks and try to use powerful report generators [6], list processors, fast logic queries [6], compiler writing system ELMA [7] as a grammatical formalism and tool for programming [8]. The so-called standardized data base schemas have proved especially useful. We have found that 3 tunable schema classes help to bring a client directly into the data processing system creation. The first schema is very simple (Fig. 1a) and has only one main record type.

The records are broken into subsets by upper structures to speed up the processing. Below there are all kinds of versions for one CASE.

The second schema (Fig. 1b) is more interesting. Here we have two tunable schemas A and B which are interleaved by some N:M relations.

It is easy to go through and add some more easily tunable schemas to A and B. For those standardized schemas we have built up specification languages, to describe data

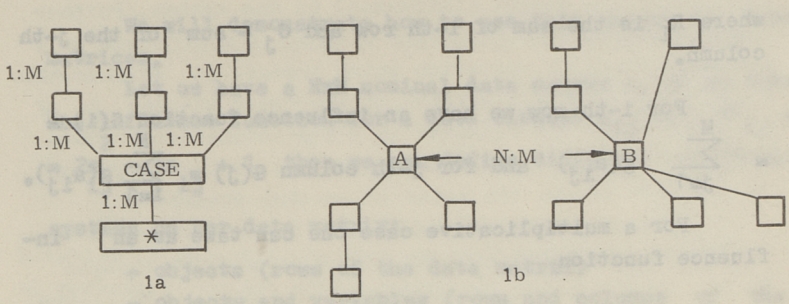


Figure 1

and the desired results with nonprocedural languages. Our main result along this line is:

> Standardized data base schemas + specification languages = automatic program generation.

We get our first implementation draft usually very quickly running. After that the appetite of our client is likely to grow and the tuning starts to make the system more effective. What is really important - the client is able to use the system from the very beginning. That is psychologically important. The client feels that he himself was creating the system and his frustrations are usually minimal.

We have developed some fairly large permanent data systems with our standard technology. F.e. cancer registers of the Estonian SSR and the Lithuanian SSR are built using this technology. The technology is applied by the Estonian Ministry for Health for a statistical system [8].

References

1. Ahmed N., Rao K.R. Orthogonal transforms for digital signal processing. Berlin, Heidelberg, New York, Springer, 1975.

2. Vyhandu L. Some problems of data analysis theory // Trans. of Tallinn Tech. Univ. 1974. No 366. P. 3-15 (in Russian).

3. Thurstone L.L. Multiple factor analysis. Chicago, University of Chicago Press. 1947.

4. Vyhandu L. a.o. Nonlinear transformation of a set of hyperspace points onto a plane // Programs for direct synthesis of models. III. Kiev, Institute of Cybernetics. 1975 (in Russian).

5. Mullat J., Vyhandu L. Monotonic systems in scene analysis // Symposium. Mathematical Processing of Cartographic Data. Tallinn, 1979. P. 63-66.

6. Vyhandu L. a.o. A system to manage and process discrete information // Control Systems and Machines. 1981. No 1. P. 99-102 (in Russian).

7. Vooglaid A. a.o. Input languages of ELMA system // Trans. of Tallinn Tech. Univ. 1982. No 524. P. 79-96 (in Russian).

8. Vyhandu L. a.o. Technology of building problem-oriented data processing systems // Trans. of Tallinn Tech. Univ. 1983. No 554. P. 13-19 (in Russian).

No 614

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED
ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

J. Tepandi, T. Luczkovsky
UDK 681.3.06

SOFTWARE SYSTEMS INSTALLATION

Abstract

A classification of software system users is given by their experience, habits, prejudices and work style. It is suggested that quality features alone do not assure the successful installation of the systems. The vendors must carefully prepare every installation step according to the users' specific needs.

1. Introduction

Every programmer thinks that he is the one who makes the best programs in the world and that he is just taking part in creating the best software system. So he hopes that his new system will have a great future and that most of the potential users are only waiting for a chance to use this system. In practice, however, it is not so fine most of the times. According to [1], very many software projects are completed, but never used. One of the reasons is that software system developers too often do not take into account the specific needs of a potential user, his requirements and wishes.

We are not pretending to solve this problem in its whole complexity. Instead, we should like to describe the experience we have had with installing our software systems [2,3] for various kinds of users around the USSR.

This paper might be useful both for those engaged in systems development and in installation and maintenance.

For clarity let us mention that by the word "installation" we mean the set of actions from unloading system on the user's site to the point when the user is able to cope with the delivered system.

2. All the user's men

During the installation process, the vendor is contacting various kinds of people. By their functions they are traditionally called as follows [4]:

- end-user who needs the information provided by the system
- data processing manager who decides which system to obtain
- application programmer who solves the end-user's problems
- maintenance programmer who helps the application programmer and changes the software when necessary.

The functions are not the only thing the vendor must deal with. The users differ also by their experience, habits, prejudices and work style. Ignoring such differences may lead to unnecessary complications. Therefore let us classify the users by these additional qualities.

Experience in automatic data processing. The users may have:

- no previous experience
- experience with another task, but not with solving this particular task
- experience in solving the same task with the aid of a different system.

Depending on the nature of his experience, the end-user may have different work habits:

- no habits
- getting the information through other people (secretary and/or application programmer)
- working online with computer.

The users may also have prejudices against/in favour of the new system:

- active willingness to use the system
- willingness to try out automatic data processing
- disinterest in installing the new software system
- active or passive unwillingness.

Moreover, the user's staff may be ready and willing to support the vendor whenever possible, or they may hope that the vendor will handle all the installation problems.

3. Preliminaries to a successful installation

Like other activities, the installation process needs planning and preparations.

Firstly, relationships between the vendor and the user must be clearly fixed. Apart from other concerns, the specific tasks before, during and after the installation must be linked to each participating person.

Furthermore, the user must be psychologically and technically prepared for the new system. If needed, the vendor must lessen the user's unfounded optimism and belief into the computers' omnipotence. If the client is pessimistic, the vendor should avoid advertising unexisting features of his system. To achieve fast and successful installation, the required preparations on the user's site should be fixed and carried out.

Secondly, the vendor must pay sufficient attention to different qualities of his system. In each case, according to the situation, he must emphasize the proper aspects of his system. Let us discuss it in detail.

4. The requirements and the users

The most important requirements to a software system are [5]: reliability, flexibility, performance, functional modularity, compatibility, ease-of-use, portability, ease-of-installation. Now let us discuss the meaning of these requirements in the installation situation.

Software reliability is always important. Usually it takes quite a while to learn it. So the user has a better attitude towards a system which has enjoyed a long maintenance practice in other organisations. If the new system has not been installed elsewhere, the vendor must assure his support for a certain period after installation.

Software flexibility is significant for the user, who is going to solve different tasks with the same system. Otherwise the flexibility might be superfluous from the user's point of view: it makes the system more complicated and diminishes its performance. Therefore, it is not always reasonable to give the user all the information about the system capabilities.

Software performance is of major importance for the

user, desiring to replace his/her old system, and for the pessimistic user. It implies the vendor must know the performance of the previous system before advertising his own one.

In many cases the performance factor is of less importance than usually presumed, especially when a new system is installed for a friendly user. Nevertheless the vendor must be ready to improve the performance if it is necessary.

Functional modularity and compatibility is of prime importance when the user has experience with another software system and wishes to detach some functions of the new system for combining them with the existing software. It gives the vendor a chance to sell the functional subsystems of his software.

Ease-of-use is highly appreciated by the end-user, who is going to work on-line. In the contrary situation, the system is chosen by those not in direct working contact, so the vendor must not overestimate this requirement.

Portability is of interest for the user, only if he is planning to replace or renew his hardware. Otherwise he hardly takes this factor into account.

Ease-of-installation enhances the user's trust in the system. It may help to overcome the unwillingness of the pessimistic user.

5. Conclusions

The quality features alone do not assure the successful installation of software systems. The vendors must carefully prepare every installation step according to the users specific needs.

References

1. Jensen R.W., Tonies C.C. Software engineering. Prentice-Hall Inc., 1979.

2. Лучковский Т.Ф., Микли Т.И., Рензер А.В. Экономические информационные системы коллективного пользования// Труды Таллинского политехнического института. 1982. № 524. С. 23-27.

3. Лучковский Т.Ф., Тепанди Я.Я., Хермлин М.П. Опыт внедрения и эксплуатации систем обработки данных - надстроек СУБД сетевого типа // Труды Таллинского политехнического института. 1983. № 554. С. 3-12.

4. Glass R., Noiseux R.A. Software maintenance guidebook. Prentice Hall Inc., 1981.

5. Gilb T. Software metrics. Winthrop Pulbishers, 1977.

No 614

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED
ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

J. Tepandi
UDK 681.3.06

DATA-DRIVEN MATRIX FORMS

Abstract

The generation of statistical accounts is discussed the contents of which are determined by two groups of conditions; one for rows and the other for columns.

The general principles are given of how to use DBMS tools to describe the accounts. The SEQUEL-2 serves as an example DBMS.

1. Introduction

Office Information Systems are becoming increasingly important both in terms of commercial products and research directions [1]. The users of an office information system are expected to be nonprogrammer professionals. It is necessary to develop a nonprocedural language as an interface between the user and the computer system. The system's language must be user-friendly so that non-experts can easily understand and use it with a minimum of training. It also must be powerful enough to provide facilities for the set-up and maintenance of a whole range of user applications [2].

The SHODI system design [4] is based on these principles. It serves as a tool for building application packages. Their most important functions are (1) data input and updating, (2) producing output forms. We shall distinguish between three types of output forms: reports, statistical reports and matrix forms.

By a report we mean a tabel which usually contains some data from the data base. The reports are formed by report generators, e.g. RPG [5]: a relational DBMS relation may be considered as such a report. Statistical re-

port is also a table, but it contains mostly derived data. The contents of such reports are determined by two groups of conditions: one condition per row and one per column [6]. In addition, there is a data selection criterion for the whole report. A matrix form is a generalization of the statical report.

The SHODI system is built on a Codasyl-type DBMS. The DBMS is used to form a data file for the statistical report generation. Each cell of the report is filled in with the number of records in the file, satisfying simultaneously the selection criterion, the corresponding row's condition and the corresponding column's condition.

Example 1. To illustrate what we mean by reports and statistical reports, consider the relational schema of Figure 1, describing a store database. The STORE relation describes a set of stores, giving the store number, branch name, number of employees in the store, and location name. The BRANCH relation gives the branch name and branch number. The LOCATION relation describes the district names and types of the locations. A fragment of the store data base is given in Figure 2.

STORE (STNO, BRNAME, EMPS, LOC)
BRANCH (BRNAME, BRNO)
LOCATION (LOC, TYPE, DINAME)

Figure 1. Example data base

STNO	BRNAME	EMPS	LOC
110	GROCERY	22	KEILA
215	STATIONERY	16	ELVA
321	GROCERY	17	ELVA
416	PERFUMERY	3	KEILA
501	GROCERY	77	NYO
508	GROCERY	25	ELVA
603	GROCERY	5	ELVA
108	DEPARTMENT-ST	45	KEILA
111	DEPARTMENT-ST	120	TOWN 1
112	DEPARTMENT-ST	150	TOWN 2

Relation STORE

BRNAME	BRNO
GROCERY	1
STATIONERY	2
PERFUMERY	3
JEWELLERY	4
DEPARTMENT-ST	0

Relation BRANCH

LOC	TYPE	DINAME
KEILA	TOWN	HARJU
ELVA	TOWN	TARTU
NYO	SETTLEMENT	TARTU
KALLASTE	TOWN	TARTU
TOWN1	TOWN	DIST1
TOWN2	TOWN	DIST2

Relation LOCATION

Figure 2. Example data base fragment

A typical report query example: "List the grocery stores in Tartu district having more than 10 employees, with their locations". The corresponding report:

STNO	LOC
321	ELVA
508	ELVA

A statistical report example is given in Figure 3. This report is determined by the data selection criterion (BRNAME ≠ 'DEPARTMENT-ST'), three row conditions (LOC = = 'ELVA', LOC ≠ 'ELVA', LOC ≠ 'ELVA' AND BRNAME = 'GROCERY') and two column conditions (EMPS < 20, EMPS > 10).

Number of Stores (excluding department stores)

	Less than 20 employees	More than 10 employees
In Elva	3	3
The rest (total)	2	1
The rest (Grocery)	1	1

Figure 3

In practical applications, there may be hundreds of different reports (usually 30 ... 100). Their sizes vary greatly, the average being about 15 rows and 6 columns.

Every condition may hold tens of comparisons. In the SHODI system, a 15x6 report description requires 1 + 15 + 6 = 22 conditions. If every cell were described separately, there would have to be 15x6 = 90 conditions. Besides, changing the report query would be a difficult task: modifying a column content would require changing of 15 cell conditions.

The real-world applications demand more powerful tools than those of SHODI. Consequently, the aim of this paper is to generalize the concept of the statistical report and its generation methods.

In the next section we will introduce the concept of a matrix form - a generalization of the statistical report. In the third section, a minimum matrix form description language (MARM) will be proposed. The last section will be devoted to matrix forms in the context of the relational DBMS; in particular, the notion of a matrix aggregate function will be introduced.

2. Matrix forms

More than twenty applications built using SHODI, show that inclusion of the following features would be highly appreciated:

- determination of the number of rows, the number of columns, the corresponding conditions and texts, that depend on the data in the data base (notice that in typical reports the number of rows is mostly determined by the data);
- determination of the hierarchical reports with rows and columns organized into a hierarchical structure (SHODI provides for imitation of the outward appearance of the hierarchical reports [7], but the content description of such a report has a linear structure - for example, a row condition is formed as a conjunction of the conditions at all hierarchy levels for this row);
- performance of the arithmetical operations on the groups of rows or columns;
- association of the conditions and the corresponding texts, for example, by using the same variable in both cases.

Example 2. SHODI is not best suited to design and maintain effectively the table in Figure 4. The main reason is that the table size depends on the data in the data base: adding a new location, for example, requires addition of a new column, and consequently changing the table query in SHODI. Furthermore, the first two rows should require a single condition (BRNAME='GROCERY'), as

The stores having more than 15 employees

		Keila	Elva	...	Total
GROCERY	NO OF STORES	1	2	...	3
	NO OF EMPS	22	42		64
STATIONERY	NO OF STORES	0	1	...	1
	NO OF EMPS	0	16		16
. . .		. . .	. . .		

Figure 4.

well as the following pairs of rows; the "Total" column requires addition of all columns; the number of employees appear both in the selection criterion (EMPS >15) and in the heading text. Finally, creation of such tables requires a tight interface with the DBMS: for example, the names of the branches may be taken from one relation (BRANCH), the names of the locations from another (LOCATION), the table may be formed using the STORE relation. To include all the enumerated features into a unified framework, we introduce the concept of the (data-driven) matrix form - a generalization of the concept of the statistical account.

Matrix form. Intuitively, we shall introduce a matrix form as a result of mutual influence (interference) of two description schemas (subtrees): the rows schema and the columns schema [8]. The first subtree's leaves define a n-element row vector, the second subtree's leaves - a m-element column vector. The n and m values may depend on data in the database.

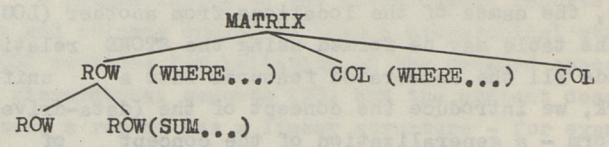
Each element of both vectors defines a relation, an operation, a text and a name. They may be determined directly for the element (i.e. for the subtree's leaf) or may be inherited from the higher levels of the subtree. Matrix form is a n x m matrix with row, column and heading texts. The texts are taken from the row and column vectors. Every cell's content is determined by the relations and operations of the corresponding row and column vectors' elements.

To define a query, several types of variables are available. Amongst them, the tuple variable (i.e. variable having tuples of a given relation as its value area) is used to describe a sequence of rows or columns - a row or column per tuple. All the variables may be used in conditions and texts.

Example 3. The query describing the form of Fig. 4 (excluding texts) is as follows:

```
MATRIX    RANGE X BRANCH
          RANGE Y LOCATION
          DATA STORE WHERE EMPS > 15
               ROW (WHERE BRNAME = X.BRNAME
               ROW
               ROW (SUM EMPS))
               COL (WHERE LOC = Y.LOC)
               COL
```

In a tree form, this description might be as follows:



Tuple variables X, Y have their values as tuples of the relations BRANCH and LOCATION, correspondingly. The form table is built using the relation STORE (DATA STORE), using the condition EMPS > 15. The descriptions given for the higher levels of the tree are carried down to the lower levels, so the relation STORE is used for all the rows and columns, adding only the complementary

restrictions (WHERE...). The restriction WHERE BRNO = = X.BRNO determines a separate row for each branch. If the operation name is not given, it is taken to be COUNT, so the description ROW determines a row to be filled with a number of tuples in this row's relation.

3. A simple matrix form description language

In principle, a matrix form description language should provide a range of powerful data definition and manipulation facilities. It would be ideal to design and implement such a language in its entirety. We did not follow this approach, because our goal was to develop an easy-to-implement language, compatible with different data manipulation systems. So we propose a minimum language to deal specifically with the matrix forms (MARM). In so doing, we suppose that, whenever possible, the traditional functions (e.g. data selection using several files) are realized by the "environment": OS, DBMS, high-level language, etc. (henceforth referred to as "the environment system"). Some ways to achieve this goal will be discussed in the next section.

The external representation of MARM is linear. It is supposed, that the linear representation and its syntax offer good tools to describe functions of a language. Furthermore, the linear query may easily be stored and modified. The subjects of the interactive user dialogue, the graphic form description and the language implementation will be presented in a separate paper.

A form description consists of the following components:

- connection with the environment system
- data description
- data manipulation and localization
- form table description
- form, row and column heading description.

We shall describe only the last two components.

Form table description. Using the DATA and WHERE clauses, a relation is associated with each row and column. Every cell is associated with the intersection

of the corresponding row and column relations.

Similarly, every cell is associated with operations for the corresponding row and column. They may be aggregate operations (COUNT, SUM, ...), row/column manipulation operations (ARIT) or READ operation.

The aggregate operations are performed first. If a cell is associated with two aggregate operations, the cell operation is selected according to the following table:

col op \ row op	COUNT	another-op
COUNT	COUNT	another-op
another-op	another-op	undefined

Cell's content is the result of performing cell's aggregate operation on cell's relation.

Firstly, all the cells, associated with two aggregate operations are filled. To indicate the order of the row or column evaluation, the row or column names may be specified in the ORDER-clause.

Form, row and column headings specify both the fixed part and the changing part of the form template. The texts are described in the TEXT-clause as strings (fixed text) and variables (changing text).

Example 4. The query for the matrix form in Figure 4:

```
MATRIX READ NOEMP
       RANGE X BRANCH
       RANGE Y LOCATION
       DATA STORE WHERE EMPS > NOEMP
       TEXT 'THE STORES HAVING MORE THAN' NOEMP 'EMPLOYEES'
           ROW (WHERE BRNAME=X.BRNAME TEXT X.BRNAME
                ROW (TEXT 'NO OF STORES')
                ROW (SUM EMPS TEXT 'NO OF EMPS'))
           COL (WHERE LOC=Y.LOC TEXT Y.LOC)
           COL (TEXT 'TOTAL')
```

In the implementation we use the experience gained

with the SHODI system and its predecessors [9, 6]. In those systems, the query is interpreted. A version of a form program generator is implemented on SM-4 computer. The generator produces a separate Fortran program for every matrix form query [10].

As there exist a whole range of algorithms for optimizing different subclasses of queries, the MARM implementation is designed to be a software tool kit rather than a "translator" or "interpretor". Every tool is best suited to some kind of queries, but they all fit in the MARM language framework. Selection of the proper tool may be performed automatically or by the user.

4. Matrix aggregate functions

In this section we shall develop an interface between the matrix form description language and an environment system - a relational DBMS. The high level of the relational languages (manipulating relations, not just records) is well-compatible with MARM - a high level specification language. We also believe that the difference between the reports and the matrix forms is best illustrated by the relational language example.

The most important differences between the matrix forms and the reports are as follows. Firstly, a matrix form has two groups of selection conditions, whilst the report is determined by one condition (or one group of conditions, or one series of conditions - if we use several relations for data selection). Secondly, a matrix form consists mostly of derived data, a report - mostly of "pure" data. In principle, every item in matrix form might be computed by means of relational DBMS aggregate functions COUNT, SUM, etc.

An aggregate function takes a relation of an argument and returns a value as a result. For example, the following query in SEQUEL 2 language [3] specifies the upper-left cell of the table in Figure 4:

```
SELECT COUNT (*)
FROM STORE
WHERE BRNAME='GROCERY' AND LOC='KEILA' AND
                EMPS >15.
```

In many relational query languages there is also a possibility to partition the tuples of a relation by equal values of a certain attribute and apply the aggregate function to each subset of tuples. Such an operation may be characterized as a vector aggregate function - a function that gets a relation and an attribute name as input and returns a vector as output. For example, a SEQUEL 2 query

```
SELECT LOC, COUNT (≡)
FROM STORE
GROUP BY LOC
```

results in the heading and the first row of the table in Figure 4. The length of the row depends on data in the STORE file.

Notice that the row or column conditions of the table in Figure 3 are not mutually exclusive. In general, a tabel of such kind cannot be specified by means of GROUP BY, it needs n x m COUNT-clauses, where n - number of rows, m - number of columns. Furthermore, the usual DBMS has no means for automated creation of forms like that in Figure 4. In real-life forms one can see both situations simultaneously: for example, a group of rows with mutually exclusive selection conditions may be followed by a couple of rows with overlapping selection criteria.

A simple n x m form with fixed n and m may be described as a matrix aggregate function with n+m+1 arguments. The arguments are a relation and n+m conditions, specified on the attributes of the relation. The result is a n x m matrix. Every cell contains the number of tuples, which satisfy simultaneously the row and column conditions. For example, in SEQUEL 2 the matrix aggregate function for the Figure 3 tabel might look as follows:

```
SELECT MATR FORM (ROW LOC≡'ELVA' LOC ≠ 'ELVA'
                  LOC≠'ELVA' AND BRNAME≡'GROCERY'
                  COL EMPS <20 EMPS >10)
FROM STORE
WHERE BRNAME≠'DEPARTMENT-ST'.
```

Notice that the data selection operation is performed by DBMS language (FROM- and WHERE-clauses).

Similarly, more complex forms may be described as matrix aggregate functions. Such a function takes one or several relations as one input, two groups (more precisely - two hierarchies) of relations as another input, and returns a matrix form table as the output.

Conclusion

A subclass of widely used office documents - matrix forms is specified. A matrix form is determined by two hierarchical specification schemas and contains mostly derived data. Both specification schemas determine a data-driven series of relations, operations and texts. The matrix form is composed as the result of interferation of these two series.

A minimum matrix form specification language is proposed, with the assumption that the traditional data description, manipulation and localization operations are performed by the environment system. An interface between this language and an environment system (a relational DBMS) is proposed to take the shape of a matrix aggregate function - a generalization of the conventional aggregate function in the relational DBMS.

Acknowledgement

The author is grateful to L. Vyhandu and T. Mikli for their support and encouragement. Careful reading of the paper by M. Laane and S. Nurk is also greatly appreciated.

References

1. Tsichritzis D. Form management // Communications of the ACM. July 1982, vol. 25, No 7. P. 453-478.

2. LuO D., Yao S.B. Form operation by example// Proc. International Conference on Management of Data. Ann Arbor, Michigan. 1981. P. 1-12.

3. SEQUEL 2: A unified approach to data definition, manipulation, and control // IBM Journal of Research and Development, 1976, November. Vol. 20. No 6. P. 560-575.

4. Vyhandu L.K., Luczkovsky T.F., Mikli T.J., Tepandi J.J. A discrete information management system // Control Systems and Machines. 1981. No 1. P. 99-102 (in Russian).

5. ES EVM. Operating System. RPG. TS. 51.804.001, 1979 (in Russian).

6. Aus T.A., Räbovõitra M.G., Tombak M.O. Matrix report generator // Proc. of the Tartu State University. 1974. No 30. P. 23-30 (in Russian).

7. Bernshtein E.B. Output of tables in SHODI system // Trans. of the Tallinn Tech. Univ. 1982. No 524. P. 39-49 (in Russian).

8. Tepandi J.J. Generation of the statistical accounts in DBMS environment. // Trans. of Tallinn Tech. Univ. 1984. No 568. P. 23-34 (in Russian).

9. Vyhandu L.K. The "SODI" architecture // Sociology of culture. Moscow, 1976. P. 339-368 (in Russian).

10. Tepandi J.J. Student team working on a contractual basis: report program generator for SM-4 computer // High School Problems. Tartu State University, 1985 (in Russian).

11. Klug A. Equivalence of relational algebra and relational calculus query languages having aggregate functions. // J. of ACM, 1982, July. Vol. 29. No 3. P. 699-717.

No 614

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED
ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

Е. Бинарин
UDK 681.3.06

STATOS - A COMPUTER SYSTEM FOR SOCIO-ECONOMIC DATA ANALYSIS

Abstract

A general description of STATOS, a computer system for the statistical analysis and handling of socio-economic data is presented. The STATOS system, the control language of the package, data management subsystem programs and the statistical analysis programs are described.

STATOS is a package for data handling and statistical analysis created at the Department of Data Processing of Tallinn Technical University and has been in use since 1978. The first version was implemented on WANG-2200 computer. The second one which is more advanced and powerful was applied on EC computer under OS/TSO. The network base management system DDMF is used for data handling and manipulation. A new version for SM-4 computer is under development.

A general description of the system will be presented here. A manual published in Russian is available [8]. STATOS has found many users in the USSR because of its flexibility, extensibility and ease of use.

1. General description of the system

One of the most important and unique aspects of STATOS is its orientation to the developed data manipulation and handling techniques. All statistical analysis programs of the system read data from a special STATOS-file created by the data management subsystem. This method has several advantages.

First, data may be listed, verified and, if necessary, edited prior to the usage. Second, the data stored in a file can be analysed by several statistical analysis programs. Eventually, new files can be created by merging, transforming and partitioning of the existing files.

Figure 1 shows a general organisational schema of STATOS.

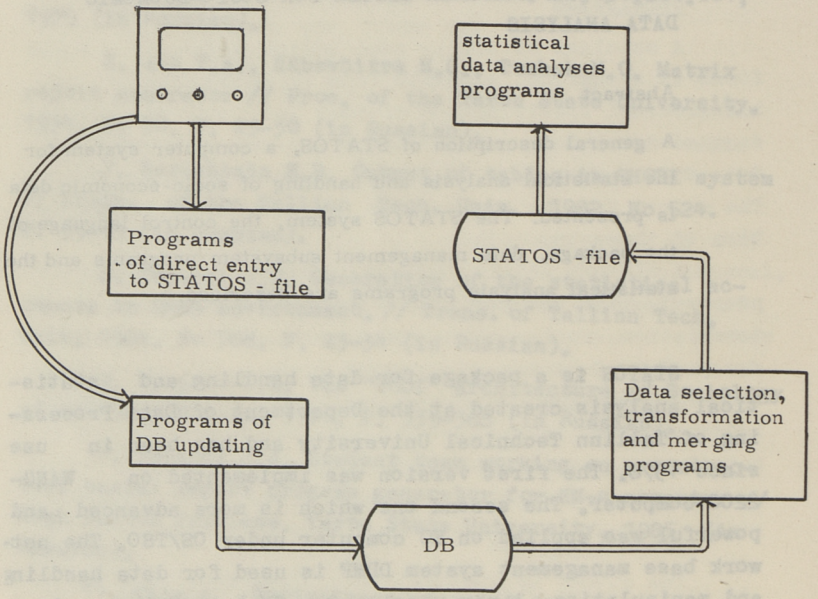


Figure 1. An organisational schema of STATOS

STATOS programs may be split into two classes:

- data management subsystem
- statistical data analysis subsystem.

The data management subsystem may be further divided into the following groups:

(1) Programs of data base management subsystem DDMF. By means of these programs a network data base is created. These programs are used by the integrated application generation system SHODI, and are hidden from the end-user.

(2) Programs of the SHODI integrated application generation system. These programs form the kernel of the data management subsystem. With these programs all data handling tasks are performed: updating, report generation, reorganisation, etc.

(3) Data selecting, transforming and merging programs in order to create STATOS-files.

The statistical programs use the data of a STATOS-file. Before statistical analysis the STATOS-file must be updated by the programs of this group.

As shown in Figure 1 STATOS can be used in two main ways:

(1) Update directly STATOS-file and then make the statistical analysis. This is the most simple, helpful and frequently used way, particularly when the number of data entries is small.

(2) Update with the help of the application generator system SHODI the data base. It is useful when the number of data entries is high, data storing and preliminary investigation are necessary.

The statistical analysis of data can be made applying one of the following programs:

SAGETO: Generation of contingency tables and calculation of table statistics

MEANST: Calculation of descriptive statistics (mean, sample standard deviation and variance, standard error of mean, maximum and minimum values of population, the coefficients of asymmetry and excess)

CORRE 1: Correlation and regression analysis

STEPRG: Multiple regression analysis

FNC 10: Nonlinear regression analysis

PWREGR: Piecewise regression

GRAFPE: Scatter plots and histograms

CLSTRT: Cluster analysis

FACTO: Factor analysis

MDISC: Discriminant analysis.

Notice should be taken of one more aspect.

One of the major principles of the implementation of the given package is its extensibility. It can be extended along different lines. First, to widen the package by new methods of statistical analysis. Second, the system of the data base used can be changed. It is achieved by using the STATOS-file as a work-file.

All those programs are quite complex and have many specific features enabling a client to get a better view of his data.

2. Systems control language

To control STATOS-package one has to use a simple key-word language to fix parameters of a given program (a list of keyword = value type terms in any order, separated by commas). Three ways are available:

(1) name of parameter = a constant

(2) name of the element of an array = a constant

(3) name of an array = a list of constants, where the constant may be either of the type of integer, real or symbol; and the list of constants is a sequence of the above mentioned constants being separated by commas.

To open that list one has to use keyword &SEDA, to close it - &END.

E.g.,

```
& SEDA  NAME = 'EXMP', IND = 1, 2, 3, 5,
M = 10, XMISS = 0., & END
```

One has to keep in mind that:

(1) The order of parameters in the language is not important.

(2) The values of parameter must be written on one line whereas the key-word and the parameter value may be written on different lines.

(3) Blank symbols are not allowed in key-words and .constants.

3. Data management subsystem

We will not describe the programs of the integrated system SHODI. This description is presented in [2]. We concentrate on programs to create, transform and merge STATOS-files.

The first of those programs FILEFM creates a STATOS-file from the data base using logical conditions and transformations. The second program THEO stores data directly into STATOS-file. Syntactic errors are found and displayed. Data elements are separated by special symbols. The third program SISE1 has the same goal, only data input is organized positionally through Format statement in FORTRAN. The fourth program FFORM selects records from an old STATOS-file by logical conditions, transforms them and creates a new STATOS-file.

There are also programs to transpose, sort and merge the given STATOS-files.

Program	Function	Brief description
FILEFM	Creation of STATOS-file from DB	Data from the data base are selected by the logical condition, transformed and written to the STATOS-file
THEO	Direct entry to STATOS-file	Data are directly stored in the STATOS file. Syntactic errors are cleared out and displayed. Special separators are used for data.
SISE 1	"	The same work is done but instead of separators data are determined positionally (Format statement in FORTRAN)

Program	Function	Short description
FFORM	Transformation of the STATOS-file	Records are selected by logical conditions from the old STATOS-file, then transformed, and the new STATOS-file is created.
COPYF	Transposition of the STATOS-file	The STATOS-file is transposed.
SFSORT	Sorting of the STATOS-file	According to the key records are sorted in the ascending order.
MESTM	Merging of the STATOS-file	Two STATOS-files are connected by the value of the key and a new STATOS-file is created.

4. Statistical analysis subsystem

When the STATOS-file is created, we may start data analysis. As the general ideas of data analysis are well-known, we describe only some of more interesting parameters or variations of implemented methods.

The program for contingency table creation generates at least 150 tables with their frequencies on one pass of STATOS-file percentages and all kind of statistics.

The program for correlation and regression analysis makes a lot to make the client's life easier. It displays maximum spanning tree of the correlation matrix and represents the set of variables in the order of their descending influence.

We want to make special mention of the program for nonlinear regression analysis FNC10 which is written along the lines of Tukey's Exploratory Data Analysis. We have added a technique to determine the form of a nonlinear function automatically.

The same program is used as a subroutine for piecewise regression where N data points are split into K classes and for every class a nonlinear curve is fitted (if needed).

STATOS has also employed several methods for high-dimensional data visualization using methods represented in [5]. For data clustering we have a rich choice of original methods based on the theory of monotonic systems.

To conclude with our system has found many followers in the USSR due to its flexibility, extensibility and ease of use.

References

1. Hartigan J. Clustering algorithms. New York, Wiley, 1975.

2. СХОДИ. Руководство программиста. Таллин, 1982.

3. НИИ культуры. Труды 32. Выпуск 3. М., 1976.

4. Тьюки Дж. Анализ результатов наблюдений. Разведочный анализ. М., 1981.

5. Выханду Л.К., Нунапуу Э.Х.-Т. Графические методы обработки социально-экономических показателей // Труды ТПИ. 1981. № 511. С. 101-110.

6. Булко И.М., Дорожко Н.Н. и др. Диалоговая система программирования - Дисп. М., 1981.

7. Сборник научных программ на Фортране. Руководство для программиста. Выпуск 1. Статистика. М., 1974.

8. Нунапуу Э. Пакет прикладных программ статистического анализа данных СТАТОС. Таллин, 1985.

No 614

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED
ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

J. Henno
UDK 681.3.01

A PRECEDENCE GRAMMAR FOR ADA

Abstract

An (1,1)-context precedence grammar for the programming language Ada is presented. Some methods for transforming grammar into the precedence form are discussed.

An Ada (1,1)-context precedence grammar is presented here. The grammar corresponds to full ADA 1983 standard [3] as strictly as ambiguities and inexactitudes present in it allow. Changes introduced for removing the drawbacks are described in another paper [2] of this volume. Methods used for removing precedence conflicts are also discussed.

Precedence grammar is one of the quickest to parse. However, the ease of parsing has to be paid for. The precedence grammars tend to be rather big (the Ada grammar to be presented contains ca 450 nonterminals and 639 productions) and are difficult to understand. Transformations into a precedence form and compile-time transformation of precedence grammar parse tree into some more economical form (abstract syntax tree, e.g. for Ada Diana [5] tree) cannot be overlooked also.

To transform the grammar into a precedence form, i.e. remove precedence conflicts a method is used here which is different from that described in [1], 5.3.4. Removing precedence conflicts involves the coding of left contexts of conflicting nonterminals in the right side of productions, using new nonterminals. To remove a conflict not only the left nonterminal from the con-

flicting pair (as in [1]), but the whole prefix of the conflicting nonterminal is renamed. For example, in Ada from the production 3.8.1 [3] incomplete_type_declaration::= incomplete_type_declaration:=

TYPE Identifier discriminant_part;

follows that the precedence relation Identifier $\doteq$; holds.

But the relation Identifier $\lessdot$; holds also, since by productions from 3.3.1, 3.4.3

full_type declaration :=

TYPE Identifier discriminant_part IS type_definition ;

type_definition := . . . derived_type_definition . . .

derived_type_definition := NEW subtype_indication

and from nonterminal subtype_indication can be derived Identificator. Which one of the relations $\doteq$,$\lessdot$ must be used depends on the left context of the nonterminal Identifier. Therefore to remove the conflict, a new nonterminal typeid is introduced:

typeid := TYPE Identificator

The right sides of productions for nonterminals incomplete_type_declaration and full_type_declaration are correspondingly changed. Such a method for removing conflicts makes the right hand sides of productions short and compiling faster.

The grammar is produced and tested, using Compiler Generating System ELMA [4] developed at Tallinn Technical University. The productions in the grammar below are mostly invertible, i.e. the right hand side of productions uniquely determines the nonterminal on the left; only 127 two-side (i.e. (1,1)) context pairs have to be remembered.

In the following, the Ada terminals are written using capital letters only, all the other words are nonterminals. Nonterminals starting with a capital letter are lexemes.

Ada simple separators are , ; . ( ) : =
< > + - & * / | '

and composite separators <> => ** /= <= << >> ..
:= >=

```
comp      := comp | comp compun
compun    := unit | withcl compun | usecl compun
unit      := prgmdcl | packdcl | gendcl | geninst |
             probd | subunit
withcl    := withid ;
withid    := WITH Ident | withidc Ident
withidc   := withid ,
usecl     := usenm ;
usenmc    := usenm ,
usenm     := USE nm | usenmc nm
subunit   := separs nm ) probd
separs    := SEPARATE (
bdystub   := prgmspec IS SEPARATE ; | taskbdid SEPARATE ;
           | packbdid SEPARATE ;
idlt      := Ident | idltc Ident
idltc     := idlt ,
idltk     := idlt :
typid     := TYPE Ident
sbtypid   := SUBTYPE Ident IS
begstms   := BEGIN seqstms
endl      := END ; | END Ident ;
end2      := endl | END String ;
idconst   := idltk CONSTANT
idnm      := idltk nm
idnme     := idnm :=
idsbtp    := idltk sbtpindc
idsbtpe   := idsbtp :=
nms       := nm (
nma       := nm'
prgma     := prgmid ; | prargass ;
prargass  := prarass )
prgmid    := PRAGMA Ident
prgms     := prgmid (
prarass   := prgms expr | prarassc expr | prarn expr
prarassc  := prarass ,
prarn     := prgms Ident -> | prarassc Ident ->
```

```
basdcl    :=  numdcl | objdcl | typdcl | prgmdcl | taskdcl |
              excpdcl | rendcl | packdcl | subtpdcl | gendcl |
              geninst
objdcl    :=  compdcl  ; | comcon | arcn ; | arcne expr ;
arcn      :=  idltk arrcnstr | idconst arrcnstr
arcne     :=  arcn :=
idconnm   :=  idconst nm
idconsbt  :=  idconst sbtpindc
idcone    :=  idconnm := | idconsbt :=
compcon   :=  dfcondef | idconsbt ; | idcone expr ;
numdcl    :=  idconst  := expr ;
typdcl    :=  flltpdcl | inctpdcl | pritpdcl
flltpdcl  :=  inctpism typdef ;
inctpism  :=  inctpis
inctpis   :=  inctp IS
typdef    :=  enumdef | realdef | recdef | derdef | intdef |
              arrdef | accdef
subtpdcl  :=  sbtypid nm ; | sbtypid sbtpindc ;
sbtpindc  :=  nm rngc | nm fconstr
fconstr   :=  dsim | dsim rngc
dsim      :=  DIGITS simexpr | DELTA simexpr
rngc      :=  RANGE rng | RANGE attr
rng       :=  simexpr  ..  simexpr
derdef    :=  newnm | newnm rngc | newnm fconstr
newnm     :=  NEW nm
enumdef   :=  enums  )
enums     :=  sl enumlit | enumsc enumlit
enumsc    :=  enums ,
enumlit   :=  Ident | String
sl        :=  (
intdef    :=  rngc
realdef   :=  fconstr
arrdef    :=  arrunc | arrcnstr
arrays    :=  ARRAY  (
arrl      :=  arrays ind | arrlc ind
arrlc     :=  arrl ,
ind       :=  nm RANGE < >
indlst    :=  arrl  ) OF
arrunc    :=  indlst nm | indlst sbtpindc
arrcnl    :=  arrays indcn | arrcnlc indcn
```

```
arrcnlc  :≡ arrcnl ,
indcn    :≡ nm | sbtpindc | rng
indcnlst :≡ arrcnl  )  OF
arrcnstr :≡ indcnlst nm | indcnlst sbtpindc
complst  :≡ RECORD compdcl  ; | complst compdcl |
            varn compdcl  ;
compdcl  :≡ dscrspec | idsbtpe expr
varid    :≡ RECORD CASE | complst CASE | varn  CASE
varpt    :≡ varid Ident IS WHEN | complst WHEN | varnn
            WHEN | varend WHEN
varnn    :≡ varn nullstmt
varn     :≡ varchlst →
varend   :≡ complst endcase | varend endcase | varnn endcase
varchlst :≡ varpt ch | varpth ch
varpth   :≡ varchlst |
ch       :≡ simexpr | OTHERS | sbtpindc | rng
recdef   :≡ RECORD nullstmt endrec | varend endrec |
            complst endrec
endrec   := END RECORD
accdef   :≡ ACCESS nm | ACCESS sbtpindc
typids   :≡ typid (
dscrspec :≡ idnm | idnme expr
discrltc :≡ discrlt ;
discrlt  :==typids dscrspec | discrltc dscrspec
inctp    :≡ discrlt  ) | typid
inctpdcl :≡ inctp ;
declpt   :≡ basdclit | bodyk | declpt basdclit |
            declpt bodyk
basdclit :≡ basdcl | repcl | usecl | prgma |
bodyk    :≡ probd | bdystub
probd    :≡ sbprgmbd | packbd | taskbd
nm       ;≡ Ident | String | selcomp | indcomp | attr
indcomp  :≡ indelst  )
indelst  :≡ nms inx | indlstc inx
indlstc  :≡ indelst ,
inx      :≡ expr | sbtpindc | rng | chlstn
chlstn   :≡ chlst → expr
chlst    :≡ Ident | chlst | Ident | String | chlst | String
selcomp  :≡ nm.Ident | nm.String | nm.ALL
attr     :≡ nma Ident | nma DELTA | nma DIGITS | nma RANGE
```

```
aggr      :≡ agg )
agg       :≡ s3 expr | chstltn expr | aggc expr
aggc      :≡ agg  ,
chstlt    :≡ s3 ch | chstlth ch | aggc ch
chstlth   :≡ chstlt I
s3        :≡ (
chstltn   :≡ chstlt →
expr      :≡ rel | relandl | relorl | relxorl | relatl | reldel
relandl   :≡ rel AND rel | relandl AND rel
relorl    :≡ rel OR rel | relorl OR rel
relxorl   :≡ rel XOR rel | relxorl XOR rel
relatl    :≡ rel ath rel | relatl ath rel
ath       :≡ AND THEN
reloel    :≡ rel oe rel | reloel oe rel
oe        :≡ OR ELSE
rel       :≡ simexprl | simexprl ni rng | simexprl ni nm |
             simexprl relop simexprl
ni        :≡ IN | NOT IN
relop     :≡ ≡ | /≡ | < | <≡ | > | >≡
simexprl  :≡ simexpr | alloc
simexpr   :≡ terml | unadop  terml
terml     :≡ term | terml addop term
addop     :≡ + | - | &
unadop    :≡ + | -
term      :≡ fact | term multop fact
multop    :≡ * | / | MOD | REM
fact      :≡ prim | prim ** prim | ABS prim | NOT prim
prim      :≡ NULL | Const | nm | aggr | qalfexpr
qalfexpr  :≡ nma aggr
alloc     :≡ derdef | NEW qalfexpr
seqstm    :≡ stmt | seqstm stmt
stmt      :≡ stm | label stm
stm       :≡ nullstmt | assnstmt | exitstmt | gotostmt |
             delstmt | raisstmt | proccall | retstmt |
             abostmt | codestmt | ifstmt | loopstmt | accstmt |
             casestmt | block | selstmt | prgma
label     :≡ << Ident >> |  label << Ident >>
nullstmt  :≡ NULL ;
assnstmt  :≡ nm :≡ expr ;
ifstmt    :≡ ifalg END IF ;
```

```
ifpt      := IF exprthn | ifpt elsifpt
exprthn   := expr THEN seqstms
elsifpt   := ELSIF exprthn
ifalg     := ifpt | ifpt ELSE seqstms
casestmt  := alternl endcase
endcase   := END CASE ;
altern    := caseexpr WHEN | alternl WHEN
casechl   := altern ch | casechlh ch
casechlh  := casechl|
alternn   := casechl →
alternl   := alternn seqstms
loopstmt  := loopbody endloop ;
itersch   := LOOP | WHILE expr LOOP | forcl LOOP
forcl     := forid nm | forid sbtpindc | forid rng
fori      := FOR Ident IN
forid     := fori | fori REVERSE
endl      := END LOOP
endloop   := endl | endl Ident
loopbody  := idltk iter | iter
iter      := itersch seqstms
block     := idltk blckdcl | blckdcl | blckbd
blckbd    := blckbody endl
blckdcl   := blckdcls blckbd
blckdcls  := DECLARE | DECLARE declpt
blcbody   := begstms exphndlr | begstms
exitnm    := EXIT | EXIT nm
exitstmt  := exitnm WHEN expr ; | exitnm ;
retstmt   := RETURN ; | RETURN expr ;
gotostmt  := GOTO nm ;
prgmdcl   := prgmspec ;
prgmspec  := procid | procfmpt | funret nm
funret    := funid RETURN | funfmpt  RETURN
procid    := PROCEDURE Ident
funid     := FUNCTION Ident | FUNCTION String
procids   := procid  (
funids    := funid  (
profml    := procids dscrspec | procids prmspec
             profmlc dscrspec | procids prmspec
profmlc   := profml ;
```

```
funfml    := funids dscrspec | funids prmspec |
             funfmlc dscrspec | funfmlc prmspec
funfmlc   := funfml ;
idmnm     := idltk mode nm
prmspec   := idnme expr | idmnm
idnme     := idmnm :=
mode      := IN | IN OUT | OUT
procfmpt  := profml )
funfmpt   := funfml )
sbprgmbd  := prgmdcpt blckbody end2
prgmdcpt  := prgmspis | prgmspis declpt
prgmspis  := prgmspec IS
proccall  := nm ;
packdcl   := packspec
packspec  := packdpt endl | packdpt privpt endl
packid    := PACKAGE Ident
packidis  := packid IS
packdpt   := packidis | packdpt basdclit
privpt    := PRIVATE | privpt basdclit
packbd    := packbdd endl | packbdd blcbd
packbod   := PACKAGE BODY
packbdid  := packbod Ident IS
packbod   := packbdid declpt | packbdid
pritpdcl  := inctpis PRIVATE ;
             inctpis LIMITED PRIVATE ;
dfcondef  := idconnm ;
rendcl    := idnm renmpt | idltk EXCEPTION renmpt |
             packid renmpt | prgmspec renmpt
renmpt    := RENAMES nm ;
taskdcl   := taskspec ;
taskspec  := taskid | taskrep endl
taskid    := TASK Ident | TASK TYPE Ident
taskent   := taskid IS | taskent entdcl
taskrep   := taskent | taskrep repcl
entdcl    := enid ; | enidr ; | enfmpt;
enid      := ENTRY Ident
enidr     := enids nm ) | enids rng ) | enids sbtpindc )
enids     := enid (
enidrs    := enidr (
```

```
enfmptl  := enids dscrspec | enidrs dscrspec |
            enids prmspec | enidrs prmspec |
            enfmptlc dscrspec | enfmptlc prmspec
enfmptlc := enfmptl ;
enfmpt   := enfmptl )
taskbd   := taskbddl blckbody endl
taskbdid := TASK BODY Ident IS
taskbddl := taskbdid | taskbdid declpt
accptid  := ACCEPT Ident
accptids := accptid (
accptind := accptids expr )
accinds  := accptind )
accfml   := accptids dscrspec | accptids prmspec |
            accinds dscrspec | accinds prmspec |
            accfmlc dscrspec | accfmlc prmspec
accfmlc  := accfml ;
accfmpt  := accptids | accfml )
accstmt  := accptid ; | accfmpt ; |
            accfmpt DO seqstms endl
delstmt  := DELAY simexpr ;
selstmt  := selwait | condentr | tmdentr
selwait  := selaltl endsel | selorpt endsel | selelse
            endsel
selbeg   := SELECT | SELECT WHEN expr →
selw     := selbeg accstmt | selw stmt
seld     := selbeg delstmt | seld stmt
selaltl  := selw | seld | selbeg TERMINATE ;
selor    := selaltl OR | selorpt OR
selorbeg := selor | selor WHEN expr →
selorw   := selorbeg accstmt | selorw stmt
selord   := selorbeg delstmt | selord stmt
selorpt  := selorw | selord | selorbeg TERMINATE;
selelse  := selaltl ELSE | selorpt ELSE | selelse stmt
endsel   := END SELECT ;
condentr := selelsc endsel
selnm    := SELECT nm  ; | selnm stmt
selelsc  := selnm ELSE | selelsc stmt
tmdentr  := selort endsel
selort   := selnm OR delstmt | selort stmt
```

```
abostmt  ::= abonm ;
abonm    ::= ABORT nm | abonmc nm
abonmc   ::= abonm ,
excpdcl  ::= idltk EXCEPTION ;
exphndlr ::= exchltn seqstms
excwh    ::= EXCEPTION WHEN | exphndlr WHEN
exchlt   ::= excwh exch | exchlth exch
exchlth  ::= exchlt I
exchltn  ::= exchlt =>
exch     ::= nm | OTHERS
raisstmt ::= RAISE ; | RAISE nm ;
gendcl   ::= gnfmpt prgmdcl | gnfmpt packdcl
gnfmpt   ::= GENERIC | gnfmpt dscrcpec | gnfmpt prmspec |
             gnfmpt pritpdcl | gnfmpt gntpdef ; |
             gnfmpt withpt ;
gntpdef  ::= inctpis (<>) | inctpis RANGE <> |
             inctpis arrdef | inctpis accdef
withprgm ::= WITH prgmspec
withpt   ::= withprgm | withprgm IS nm | withprgm  IS <>
gninst   ::= packidis newnm | prgmspis newnm | funid is
             newnm
geninst  ::= gninst ;
repcl    ::= forattr simexpr ; | forid aggr ; |
             forid AT simexpr ; | recrepcl
forid    ::= FOR Ident USE
forattr  ::= FOR attr USE
recrepcl ::= recrep endrec ;
aligncl  ::= AT MOD simexpr ;
compcl   ::= nmatsmex rngc ;
nmatsmex ::= nm AT simexpr
recrep   ::= forid RECORD | forid RECORD aligncl |
             recrep compcl
codestmt ::= qalfexpr ;
```

References

1. Aho A.V., Ullman J.D. The theory of parsing, translation and compiling, vol I. Prentice-Hall, 1972.

2. Henno J. Some remarks on Ada RM grammar // In this volume.

3. The programming language Ada Reference Manual // Lecture Notes in Computer Science. 1983. No 155.

4. Vooglaid A., Lepp M., Liib D. Metalanguages for the system ELMA // Trans. of Tallinn Tech. Univ. 1982. No 524. P. 79-96 (in Russian).

5. Uhl J., Drossopoulou, Persch G., Goos G., Dausmann M., Winterstein G., Kirchgässner W. An attribute grammar for the semantic analysis of Ada // Lecture Notes in Computer Science. 1982. No 139.

No 614

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED
ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

J. Henno

UDK 681.3.01

SOME REMARKS ON ADA REFERENCE MANUAL GRAMMAR AND SYNTAX

Abstract

Inexactitudes and ambiguities present in Ada Reference Manual syntax grammar are discussed. Alternative constructs to improve the grammar are presented. Alternatives are also suggested to remove some undesirable features of Ada syntax (nonuniform syntax of semantically similar constructs, etc.).

Syntax of programming languages and methods of syntax presentation and analysis have been thoroughly investigated. Usually language syntax is defined in two steps: first, the context-free features are introduced, using a context-free (Backus-Naur) grammar or, equivalently, syntax charts, etc., and then the context-dependent restrictions are presented. Compile-time syntax control follows the same pattern. Syntax analyser (parser) checks programs correspondence to the context-free syntax grammar of the language. Correctness of context-dependent features is investigated later. Thus usually even by the word "syntax" only context-free features of syntax are understood.

Since the pioneering works of J.W. Backus and P. Naur it is commonly agreed that the abstract, succinct form of grammar greatly improves syntax presentation and makes it easier to avoid different implementations. Unfortunately, up to now no good formalisms for presentation of syntax context-dependent features have been found. The only widely known attempt to give an abstract

presentation of the whole syntax was the Algol-68 syntax presentation by van Wijngaardens two-level grammar [6].

Syntax analyser is derived directly (in compiler-generating systems automatically) from the syntax grammar. Therefore the languages Reference Manual (RM) grammar should obey some nearly obvious conditions:

a. The whole syntax must be presented in a precise, concise way. There should be no need for additional verbal clarifications about the context-free features of syntax, scattered around in RM. Useless nonterminals, which do not occur on the right hand side of productions should be excluded.

Besides syntax analysis parser constructs its parse tree (derivation tree). Since this tree reflects the semantics of the program, unambiguity is of utmost importance, i.e. for every syntactically correct program only one parse tree is allowed. In terms of grammar:

b. RM grammar must be unambiguous.

Syntax grammar is also used to learn the language syntax and to check for syntactic errors, thus

c. Grammar must be user-oriented, nonterminals system of the grammar must be as simple as possible and chosen according to the language semantics.

d. The syntax of the language must be designed to help the user - (semantically) similar constructs should have similar syntax, no exceptions are allowed.

For compiler-writers the following features are also desirable:

e. RM grammar should belong to some well-known class of grammars to allow quick and simple methods of analysis. Nowadays LR(I) (LALR(I)) or LL(I) grammar seems to be most preferable.

f. The language syntax should be as context-free as possible, i.e. context sensitive features of syntax should be avoided if possible. This makes the language easier to use and compilers far more efficient.

The above features are somewhat contradictory.

A complete syntax description may involve some rather incomprehensible and awkward constructs in grammar. For instance, in Ada several nonterminals are described as nonempty lists, where several choices for every item are possible (variant_part, actual_parameter_part). But some restrictions are present on the occurrence of choices, e.g. after occurrence of some kind of choices only some others are allowed (in procedure parameter passing list positional and named parameters are allowed, but once a named parameter is used the rest must also be the named ones). These restrictions can be expressed by context-free productions, but it is rather difficult to understand the result if the general structure of productions is left unchanged, i.e. these exceptional items are included in the list. Since the restrictions are logical and natural, they can be expressed verbally, as done in Ada RM. A better solution would be restructuring of productions removing these exceptional items from the list.

Restricting the grammar to belong to some definite class of grammars may also involve difficulties, especially when some class of grammars different from LR(k) is considered desirable. The precedence grammar produced at Tallinn Technical University for Ada [1] contains ca 500 nonterminals and ca 650 productions and these figures cannot be essentially decreased; clearly such a monster is not humanly understandable and acceptable.

However, the above features should be present in every RM grammar as completely as possible, but this is not the case with Ada RM grammar [3]. Many problems, discussed earlier [2,5] for Ada 1980 version [4] are still unsolved, and even some more have come up.

The first question arises already on your first glance at the grammar. The grammar is preceded by a remark stating that "syntax summary is not part of standard definition of Ada programming language". Does this mean that Ada syntax must be inferred from the verbal text and the examples above?

a.1. The grammar does not specify the whole syntax. From RM text it follows that attribute identifiers include reserved words (grammar terminals) DELTA, DIGITS, RANGE to be taken into consideration in 4.1.4 (further all the paragraphs will be from grammar in [3]).

a.2. Nonterminal pragma does not occur in the right hand side in any production, i.e. the grammar does not specify the occurrence of pragmas. In RM it is stated that pragmas are allowed only "after a semicolon delimiter, but not within a formal part or discriminant part and at any place where the syntax rules allow a ... category whose name ends with "declaration", "statement", "clause" or "alternative" or one of the syntactic categories variant and exception handler, but not in place of such a construct; also at any place where a compilation unit would be allowed. Additional restrictions exist for the placement of specific pragmas".

This passage is far longer than the one from ADA 1980 RM and involves more problems (which are to be solved by language designers rather than by implementors). The straightforward implementation of this definition would imply a good deal of ambiguities e.g., in the production for nonterminal basic_declarative_item from 3.9 pragma can be reduced to basic_declaration, representation_clause or use_clause - which one is correct? The first sentence of this definition is also superfluous, since all constructs with semicolon as endsymbol are presented in the following. A solution to the placement of pragmas will be given below.

1. Pragma is introduced as an alternative of component_declaration, i.e. the corresponding production from 3.7 is changed to component_declaration ::=

identifier_list : subtype_indication :=expression ;
| pragma

2. In variant part (3.7.3) pragmas can appear implicitly only at the beginning of the list of variants; after the first variant pragmas are reduced to component declarations of the previous variants of the component

list to avoid ambiguities. Thus the first production of 3.7.3 has to be changed into

variant_part ::=

```
CASE ident IS
   [pragma]
    variant
   {variant}
END CASE ;
```

3. Pragmas must be added as an alternative of basic_declarative_item, thus the corresponding production from 3.9 would be basic_declarative_item ::= basic_declaration

| representation_clause | use_clause | pragma

4. The production for nonterminal later_declarative_item is ambiguous (see b.8 below). Thus instead of there pragma should be added to the production for body from 3.9

body_ ::= proper_body | body_stub | pragma

Notice that there must be at least one body before pragma in the list of basic and later declarative items, otherwise parsing of pragmas in the declarative part is ambiguous.

5. Pragma must be an alternative of nonterminal case_statement_alternative, thus the corresponding production from 5.4 should be

case_statement_alternative ::=

```
WHEN choice {choice} ⇒
  sequence_of_statements
pragma
```

6. In task specification (9.1) both the entry declaration and representation clause can be replaced by pragma, which makes parsing of pragma between them ambiguous. This is another syntactical construction, which is rather troublesome to describe by context-free productions and should therefore be avoided by language designers. It can be done by adding pragma as an alter-

native to the representation clause, changing the corresponding production from 13.1 to

representation_clause ::=

type_representation_clause | addres_clause | pragma

and introducing a new nonterminal entry_list by

entry_list ::= entry | entry entry_list | pragma entry-list

so that task_specification would become (9.1)

task_specification ::=

```
TASK TYPE identifier  IS
     entry_list
     representation_clause
END  identifier
```

7. Pragma must belong as an alternative to nonterminal select_alternative, i.e. the corresponding production from 9.7.1 should be

select_alternative ::=

```
WHEN condition
     select_wait_alternative
pragma
```

8. By the above definition pragma is allowed at any place where exception_handler is permitted, but not in place of it. Since exception handlers form everywhere lists that probably means that such a list must contain at least one exception handler among pragmas. This is again rather awkward to describe. These lists should be changed everywhere to the following construct:

```
pragma
exception_handler
pragma  exception_handler
```

9. Since pragma could replace a statement, in 5.1 the corresponding production should be changed into

statement ::= [label] simple_statement | [label] compound_statement
| pragma

10. Pragmas may appear as part of context clause, preceding a compilation unit or replace a whole compila-

tion unit in the list of compilation units (10.1). This again makes parsing of pragma ambiguous - it can be reduced either to standing alone compilation unit or to the context clause of the next compilation unit. In the following solution only the latter is allowed: change the production for context_clause in 10.1.1 to
context_clause ::= with_clause | use_clause | pragma

a.3. Nonterminals logical_operator and highest_precedence_operator in 4.5 are useless and the corresponding productions should be removed from the grammar.

a.4. From the nonterminal declarative_part an empty string can also be derived by 3.9. Thus by 5.6 it is allowed that in block statement word DECLARE is not followed by any declaration. Introducing such meaningless words is undesirable, but can be easily corrected. Only productions for nonterminals declarative_part and context_clause allow derivation of an empty string. Since Compiler Generators often do not allow such productions, it would be better to exclude such a possibility also from RM grammar. For nonterminal declarative_part it is especially logical, since it is everywhere used as optional, i.e. here empty string is also ambiguous.

The word DECLARE is used only before block declarations, in all other constructs (package, subprogram, task) declarations are not preceded by it, what clearly contradicts to the language syntax design principle e. above. It would be better to omit the word also from block (or use it everywhere). This would not imply any new difficulties and the syntax would become more homogeneous.

a.5. In some places grammar could be made more exact, e.g. in the productions for exit (5.7) and goto (5.9) statements instead of name identifier should be used. RM explicitly states that these statements could not transfer control out of a given program unit. Like-

wise, in productions for procedure_call_statement and function_call_statement (6.4) identifier or selected_component should be used instead of procedure_name, function_name. This holds also for nonterminals generic_package_name, generic_procedure_name, generic_function_name in productions for nonterminal generic_instantiation (12.3) and these replacements remove some ambiguities from the grammar (see below).

The grammar is ambiguous in many places.
b.1. A big problem is parsing strings
name (name {, name}) (1)

Nonterminals indexed_component, slice and function_call all may look like (1) and have the same context, since by 4.1, 4.4
name ::= . . . | indexed_component . . . | slice . . .
primary ::= . . . | name . . . | function_call . . .

Also nonterminal subtype_indication may appear as (1), if the constraint in 3.3.2
subtype_indication ::= type_mark constraint
is either index_constraint or discriminant_constraint and again context does not help, since by 3.3.2
type_mark ::= name
thus it is impossible to differentiate between the constraint and typemark (i.e. name).

Apparently the best solution would be to distinguish between these alternatives by the use of different syntactic notations, e.g. different kinds of brackets [5]. If the syntax is left unchanged, a new nonterminal must be introduced for every case above and the nonterminals name and subtype_indication correspondingly changed, e.g. subtype_indication presents only alternatives with range, fixed- or floating-point constraints.

b.2. Similarly, in procedure_call_statement (6.4) it is impossible to say whether the list in brackets is actual_parameter_part or belongs to procedure_name (if all actual parameters are missing, i.e. parameter association is entirely by position). However, a procedure_name cannot contain brackets and can be either identifi-

cator or selected_component only. In this way a more specific grammar results which removes the ambiguity.

b.3. Nonterminal "entry_call" (9.5) can be distinguished from "procedure_call" (6.4) only semantically therefore it must be removed from the grammar.

b.4. In parsing generic_instantion (12.3) it is impossible to say whether the list in brackets is generic_actual_part or part of generic_package_name (generic_procedure_name, generic_function_name). This ambiguity can be removed either by some kind of "syntactic sugar" or by allowing generic_procedure_name and generic_function_name nonterminals identifier or selected_component only in place of generic_package_name.

b.5. Parsing a string

name'ident (expression)

is also ambiguous. Such a string can be reduced to name either by name

::= attribute ::= prefix'attribute_designator

prefix ::= name

attribute_designator ::= identificator (expression)

(see 4.1, 4.1.4) or by

name ::= indexed_component ::= prefix (expression)

prefix ::= name ::= attribute

attribute ::= prefix' attribute_designator

attribute_designator ::= identificator.

Expressions in parantheses which belong to attribute_designator could be distinguished using different brackets.

b.6. Nonterminal deferred_constant_declaration (7.4) may look like object_declaration (3.2) and both are basic_declarations (3.1), i.e. they cannot be separated syntactically. To make a difference, object_declaration must be made more exact, and if the word CONSTANT appears in object declaration, initialization must be present; uninitialized constants are deferred constants.

b.7. Many ambiguities contain alternatives for nonterminal primary (4.4).

Nonterminal string_literal must be removed from the list of alternatives since it can be reduced to name

through operator_symbol (6.1, 4.1).

Nonterminals function_call and type_conversion may look like name (see above), thus either their syntax is changed or they are removed from the alternatives of nonterminal primary.

Nonterminal aggregate may look like another alternative (expression), thus either its syntax is changed or only aggregate is left as an alternative of primary. In case it has the form (expression), a semantical subprogram of parser must analyze its meaning.

b.8. Nearly all the alternatives of nonterminal later_declarative_item are also alternatives of the nonterminal basic_declarative_item (see 3.9). Introduction of these two nonterminals seems to follow only didactic purposes. To avoid ambiguities later_declarative_item should be removed from the grammar and nonterminal declarative_part (3.9) defined by production

declarative_part ::= basic_declarative_item body

b.9. Introduction of nonterminals library_unit and secondary_unit (10.1) also looks like an attempt to explain semantics of the language by means of context-free syntax grammar. Only semantical analysis allows to determine whether subprogram_body is to be reduced to library or secondary unit. To remove the ambiguity these two nonterminals should be replaced by one, which would cover all their alternatives.

There are other questions related to nonterminals system and structure of the grammar.

c.1. Why besides nonterminals name and identifier is introduced nonterminal simple_name, which by 4.1 is always an identifier?

c.2. Nonterminal component_subtype_indication, which according to 3.7 is simply subtype_indication is also non-essential.

c.3. Similar constructs should be described by similar productions - this makes it easier to read the grammar. For instance, it would be better if productions

describing unconstrained_array_definition and constrained_array_definition were similar.

c.4. Italicized prefixes should be part of the grammar, i.e. instead of explaining that they are to be ignored it would be better to add the corresponding productions - implementors have to do it anyway.

d.1. Because of ambiguities Ada grammar does not belong to any parsing-efficient subclass of context-free grammars. Even after removing ambiguities more than two symbols look-ahead is needed in some places (i.e. it cannot be directly transformed into an LR (1) grammar, see below).

e.1. All other identifier declarations (object declaration 3.2, component_declaration 3.7, discriminant_specification 3.7.1 etc.) start with identifier_list, but renaming_declaration (8.5) must start with only one identifier. If similar objects can be declared together, why can they not be renamed together? Such irregularities involve difficulties not only for the user, but also for a parser implementor. Nonterminal identifier_list must be split into two: one for standing alone identifier and the other consisting of at least two identifiers; all productions containing nonterminal identifier_list must be changed accordingly. Sufficient (more than two symbols) look_ahead would be another solution to determine whether standing alone identifier must be reduced to an identifier_list (beginning with object, etc. declaration) or not (beginning with the renaming declaration).

e.2. Nearly all type declarations follow the pattern (see 3.3.1) TYPE identifier discriminant_part IS type_definition, where the first symbol derived from nonterminal type_definition determines the type in question (record, array, etc.), but task type definition begins for some reason TYPE TASK ... .

e.3. Nonterminals actual_parameter_part (6.4) and generic_actual_part (12.3) are very close seman-

tically and have also rather similar syntax. There is only difference - for generic_formal_parameter it is allowed to be also string literal. But in RM grammar syntax description the similarity is rather difficult to understand, RM productions

actual_parameter ::= expression variable_name
type_mark(variable_name)

generic_actual_parameter ::= expression variable_name
subprogram_name entry_name type_mark

both reduce after removing syntactical ambiguities to the same

actual_parameter ::= expression

generic_actual_parameter ::= expression

The syntax were easier to remember, if nonterminals actual_parameter_part and generic_actual_part had the same syntax.

e.4. Differences between syntax of nonterminals parameter_specification (6.1) and the first alternative of generic_parameter_specification (12.1) are also difficult to remember. In generic_parameter_specification it is not allowed to use alone mode OUT and semicolon character is used as endsymbol, but for parameter_specification as delimiter, i.e. is missing after the last one (in Ada 1980 version they had the same syntax).

The suggestions made above for changing Ada syntax (adding some syntactic "sugar") would make it more context-free and thus Ada compilation faster.

Here are some more suggestions about Ada syntax.

Word IS in type declarations (see above) makes them more human, easier to read and understand. This feature would be good in object declarations too: instead of

```
A, B : T;
C : CONSTANT TI := CI
```

more readable would be

```
A, B : VARIABLES OF TYPE T;
C : CONSTANT OF TYPE TI := CI;
```

or even

A, B ARE VARIABLES OF TYPE T;
C IS A CONSTANT OF TYPE TI EQUAL TO CI;

etc.

The last example indicates, that the syntax suggested would be varying. But this is a feature usually present in good translators - some delimiters could be missing, word order cannot be significant, etc. Ada is semantically rich, but its syntax is poor and rigid and since it is the policy of Ada not to allow any dialects, it would be better to introduce some freedom into its syntax officially.

Ada, being a major development in "big" languages (which are needed for big applications) is often criticized along the lines "small is beautiful". But big can also be beautiful if it is well-organized and systematic. Hopefully some of the above remarks will be helpful for new Ada standard developers or for designers of other languages.

References

1. Henno, J. A precedence grammar for Ada (in this volume).

2. Machanick P. A note on C.S. Wetherell's "Problems with the Ada reference grammar" // SIGPLAN Notices. 1985. 18, No 5. P. 44–45.

3. The programming language Ada reference manual // Lecture Notes in Computer Science. 1983. No 155.

4. The programming language Ada reference manual // Lecture Notes in Computer Science. 1981. No 106.

5. Wetherell C.S. Problems with the Ada reference grammar // SIGPLAN Notices. 1981. 16, No 9. P. 90–104.

6. Wijngaarden, van A. Recursive definition of syntax and semantics // Ed. T.B. Steel. Formal description languages for computer programming. Amsterdam, North-Holland, 1966.

No 614

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED
ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

D. Liib
UDK 681.3.01

A TECHNOLOGY FOR BUILDING A COMPILER-WRITING SYSTEM

Abstract

Two aspects related to automatic production of a compiler-writing system are discussed. The use of the compiler-writing system for programming with grammar formalism, and optimization of memory requirements for precedence grammars are described.

Computer-aided technologies for programming tools are one of the major concerns in software development. We will discuss problems related to the automatic production of a compiler-writing system. The main aim is to build a compiler-writing system for practical use and serial production. The basic aspects involved are:

- examination of the compiler-writing system for programming with grammar formalism as immediate programming tools for classes of structured data processing tasks
- minimization of memory requirements for precedence grammars using (1,1)-bounded canonical context [7].

To use grammar formalism as a part of high-level programming languages, the concept of a tree-structured variable is introduced [5]. This approach enables us to describe an analytic calculation system [9] and a syntax-oriented input generator for net structured data bases [6].

The minimization of memory requirements consists of methods and algorithms that permit to convert the

initial precedence grammar into the precedence grammar for which two linear precedence functions exist [11]. The types of conversion that change the structure of the precedence grammar rules also change the direction of the arcs in the linearisation graph, if the graph is cyclic. The reason for the absence of the precedence functions for the initial precedence grammar is eliminated by these types of conversions.
Consequently, the information in the precedence matrix can be represented by a pair of precedence functions. The matrix consists of n^2 entries and two functions contain $2n'$ entries, when $n' \approx 4n$. Such a replacement evokes the following problems:

- context quantity minimization for reduction of precedence grammar rules with equal right sides
- restoration of the recovery of syntactic errors in time.

Tree-structured variable. To connect grammar formalism with high-level languages, the concept of the tree-structured variable is introduced. A tree-structured variable like a common variable in programming languages can be described by three attributes: name, type and value.

In the declaration of a common variable the name is attached to a given description of the data type. But in case of a tree-structured variable, the name is fixed with a number of data type descriptions. This description is achieved by describing the rules that are similar to the grammar formalism. To assign a value to common variables, first, if needed, the conversion of the type of the assigned value is made and after conversion the name is connected to the value. With a tree-structured variable its initial value is changed to the form of a tree and the name connected to the root of this tree.

The approach described above has been used in creating the analytic calculations system [9] . For example, the package for differentiation has been implemented. The initial expression is converted into

an expression, which is the derivative of the initial expression. The conversion is followed by the calculation of the derivative.

The declaration of the tree-structured variable as a universal data input system [6] is carried out in three stages:

- the definition of logical data base schema as a declaration of the tree-structured variable
- the description of the input data as a given value of the tree-structured variable
- data selection from data base.

Each given value of the tree-structured variable must be a data base subschema. Thus the relation between the structure of the input data and the logical network of the data base is determined automatically, and on these bases the access path to a record in the network is uniquely determined.

The minimization of the syntactic parsers. Here we will deal with the minimization of such syntactic parsers that work in the class of precedence grammar using (1,1)-bounded canonical context reduction [7] . The use of this method in the system ELMA implies one more conversion in the following sequence.

First the user writes his language in the form of the regular context-free grammar. Further the context-free rules are automatically transformed into a precedence grammar and it is verified whether the obtained precedence grammar is reducible using (1,1)-bounded canonical context or not. This verification is done when grammar includes rules with equal right sides. During the syntactic parse (during reduction phase), one of the appropriate rules must be chosen. To achieve such a choice it must have a set of (1,1)-context (for each rule with equal right side every possible pair of single symbol on the left and right). Eventually, the conversions result in a grammar which is transformed into the precedence grammar with two precedence functions.

At this time in view of practice no effective

methods are available for automatic transformation of a precedence grammar for a grammar for which two precedence functions exist. The author has employed a new method based on the approach of Martin [14] and Babitshev, Pronina, Trahtengerts [2].

In Martin's paper [14] it is proved that linear precedence functions exist if and only if linearisation graph is acyclic. This graph is a partially directed graph. First the partially directed graph must be transformed into a graph with the maximum number of directed arcs (the undirected arc corresponds to the precedence relation $\doteq$ in the precedence matrix), on condition that the number of cycles stays unchanged. This complemented graph orientation is carried out straight on the precedence matrix. The changed precedence relations $\doteq$ are divided into three groups and are marked as follows:

$$\doteq = \doteq^{(<)} \cup \doteq^{(>)} \cup \doteq^{(<,>)} .$$

The precedence relation $\doteq$ related to groups $\doteq^{(<)}$ or $\doteq^{(>)}$ when replacing $\doteq$ with $>$ or $<$ the arc is oriented in the linearisation graph on condition that the number of cycles remains unchanged. To the group $\doteq^{(<,>)}$ are related those precedence relations $\doteq$ which do not adjust even under additional orientation uniquely.

The acyclic linearisation graph is obtained with the help of the stratifications to the left and to the right in the precedence grammar rules, so that we can change the direction of the required arcs.

In [11] it is proved that those stratifications to the left and right make the linearisation graph acyclic and the transformed grammar is a precedence grammar.

Up to here the linear precedence functions have been observed separately from the syntactic parser. These modifications which bring along the replacement of the precedence matrix with linear precedence functions in the syntactic parser will be discussed further.

Between any pair of grammar symbols there are four precedence relations: $<$, $\doteq$, $>$ or "blank" (no pre-

cedence relation between the corresponding symbols). The "blank" entries in the matrix are not reflected in the linear precedence functions. The correspondence between precedence matrix M and linear precedence functions f and g is the following:

$M(X,Y) = \gtrdot$ implies $f(X) > g(Y)$
$M(X,Y) = \doteq$ implies $f(X) = g(Y)$
$M(X,Y) = \lessdot$ implies $f(X) < g(Y)$.

Consequently, both the possibility of recovering syntactic errors in time and that of using class K_2 for reduction phase are lost. Those limitations reduce the efficiency of syntactic parsers. To improve the efficiency of syntactic errors recovery the concept of partial equivalence of syntactic parsers is introduced. The rules of precedence grammar are divided into three classes: K_1, K_2, K_3 to improve the efficiency of reduction phase.

Cascade technique is used for classes K_1, K_2, K_3. Class K_1 consists of rules with different right sides. Classes K_2 and K_3 have rules with equal right sides. To reduce those (1,1)-bounded canonical context [7] determined on the precedence matrix is applied because the use of class K_2 and class K_1 is equally efficient. The ordered vectors have been introduced for restoring class K_2.

Ordered vectors. For all rules with equal right sides an interval between the internal codes of the left and right context symbols is indicated. If the internal code of a current symbol belongs to the corresponding internal code during the syntactic parsing, then the right rule is found. If it is not the case, the next rule with the same right side is observed.

Reflective-linear ordering between the context symbols is necessary to achieve the intervals described above. In fact, it is possible to construct an example, in which the reflective-linear ordering of context symbols does not exist. In [12] it is proved that stratifications

on the rules of precedence grammar into the precedence grammar with two linear precedence functions (to transform the initial precedence grammar into the precedence grammar with two precedence functions) are capable of changing the appearance of the symbol in the (1,1)-bounded canonical context. Those stratifications lead to the reflective-linear ordering of context symbols.

Error detection in the precedence parsers for precedence grammar has two precedence functions. In the paper by Aho and Ullman [13] the syntactic errors detection has been studied. They give the division of "blank" entries into two groups. To the first group belong only those entries that are never consulted during the parse of any input. The second group involves all the other "blank" entries. For that reason it is necessary to restore all "blank" entries for error detection in time. The restoring of those "blank" entries does not mean that errors remain undetected, but the probability of the recovery is reduced.

To compromise between minimization of memory requirements of parsers and syntactic errors detection in time the concept of simple equivalence of precedence parsers is introduced. Two parsers are equivalent if they accept the same set of inputs and print an error statement in the same point of the input set. Observation of this condition allows to the second parser only some reduction phases, when the first prints out an error statement.

For a precedence grammar using (1,1)-bounded canonical context reduction, the partial equivalence of precedence parsers is introduced. To achieve the partial equivalence of parsers some changes in the precedence matrix must be made prior to computing of linear precedence functions and an additional matrix for terminal symbols [10] must be formed.

Practical results. Experience of finding ordered vectors and linear precedence functions has been obtained by the use of corresponding algorithms in the framework of the compiler-writing system ELMA [4]. The input

languages of the system PARES [8], the metalanguages of the system ELMA [4], the syntactic parser of high-level programming language ADA [3] and a number of problem-oriented languages have been implemented.

In conclusion, the results of transformation of the context-free grammar into a precedence grammar with linear precedence functions are shown in the table below.

name of language	before transformation			after transformation	
	number of rules	number of symbols	memory for preced. matrix (Kb)	increment of rules	memory for linear prec. func., ordered vec. and add. matrix (Kb)
1. ADA	620	459	75,2	+96	4,8
2. ATRSEM	153	153	6,1	+5	1,6
3. DAMAL	521	440	47,3	+4	3,5
4. ELMAGUIDE	75	70	1,6	+10	0,6
5. ELMAMETA	183	176	7,6	+10	1,6
6. KLARA	22	25	0,2	+1	0,2
7. TRADEL	120	107	3,0	+3	0,8
8. TRODIK	101	114	2,6	+3	0,9

References

1. Aho A., Ullman J. The theory of parsing, translation and compiling. Vol. I: Parsing. Prentice-Hall, Inc., Englewood Cliffs, N. J. 1979.

2. Babitshev A., Pronina V., Trahtengerts E. Context-free grammar for which precedence functions exist // Programming. 1977. No 3. P. 43-53 (in Russian).

3. The programming language Ada reference manual // Lecture Notes in Computer Science. 1983. No 155.

4. Vooglaid A., Lepp M., Liib D. Metalanguages of the system "Elma" // Trans. of Tallinn Tech. Univ. 1982. No 524. P. 79-96 (in Russian).

5. Vooglaid A., Liib D. Grammar formalism as immediate tool for modular system programming // Theses. All-Union Seminar on Methods for Synthesis of Standardized Modula Systems. Moscow, 1981. P. 77 (in Russian).

6. Vooglaid A., Liib D. Structured tool for data processing description and realization // Theses. Automatized production of packages of application programs. Tallinn, 1980. P. 41-43 (in Russian).

7. Vooglaid A., Tombak M. Some reduction problems in precedence grammar // Trans. of Tallinn Tech. Univ. 1975. No 386. P. 23-27 (in Russian).

8. Kracht W., Eivak J. Data manipulation language DAMAL. Tallinn, 1982 (in Russian).

9. Liib D. Instrumental tool for analytical calculations // All-Union Conference on Compiling Methods. Novosibirsk, 1981. P. 162-164 (in Russian).

10. Liib D. Effective methods for programming languages realization. Theses. Tallinn, 1984. P. 15 (in Russian).

11. Liib D. About conversion of the precedence matrix to the precedence functions // Trans. of Tallinn Tech. Univ. 1983. No 554. P. 99-109 (in Russian).

12. Liib D. Optimization of memory requirements for precedence grammars using (1,1)-bounded canonical context reduction // Trans. of Tallinn Tech. Univ. 1984. No 558. P. 81-89 (in Russian).

13. Aho A., Ullman J. Error detection in precedence parsers // Mathematical Systems Theory. 1973. Vol. 7. No 2. P. 97-113.

14. Martin D. A Boolean matrix method for the computation of linear precedence functions // Comm. of the ACM. 1972. Vol. 15. No 6. P. 448-454.

No 614

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED
ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

M. Lepp, A. Vooglaid, L. Vyhandu
UDK 681.3.06

ELMA AS AN INSTRUMENTAL SYSTEM FOR BUILDING PRACTICAL PROGRAMMING SYSTEMS

Abstract

The instrumental system ELMA is described, consisting of two subsystems: a compiler writing system and a system assuring navigation between users' actions. The latter is based on Jackson's structured programming ideas. These two subsystems quarantee certain functional capabilities also discussed here.

The Department of Data Processing concurrently with a team from the Computing Center of Tallinn Technical University has worked in the field of compiler writing systems building over the past fifteen years. As a result, we have now the fourth version of our system ELMA. Our views on programming have evolved step-by-step to a more general treatment and now ELMA can be used as

- a system to ease program package writing using grammar formalisms as very high-level programming tools
- a compiler writing system.

ELMA consists of two main subsystems, the first (P1) being a compiling metasystem and the second (P2) - an interpreting metasystem to navigate between users' actions.

Both subsystems have the corresponding meta-languages - ELMAMETA and ELMAGUIDE. To realize a problem-oriented system we have to describe system languages lexical, syntactical and semantical levels in ELMAMETA.

Subsystem P1 generates language-dependent parts of the target compiler from those descriptions. The language ELMAGUIDE determines the configuration of the compiler and the connections with the operating system. If a description of the source language has been supplied, target compilers with different characteristics can easily be generated.

ELMA as a compiler writing system

We will represent the process of the description and realization of a given source language by stages.

A. Abstract syntax of the source language is described as a set of regularized context-free grammars.

B. Attribute technique is used to describe semantics of the language. The semantic actions and attributes are directly connected with the abstract syntax. We have built up a theory of abstract attribute grammars [1] and implemented special tools (a generator of transformations from given syntax into an abstract one; a generator for evaluator, which uses synthesized iterative and simple attributes, initialized inherited attributes (the evaluation of which can be planned statically) and global attributes.

C. Semantic actions are debugged separately using an interactive debugger which simulates abstract syntax and attribute environment.

D. Concrete syntax is described and debugged.

E. Code generator description is given. To generate a code we can use a simple syntax directed translation schema which is an addition to ELMA, implemented through FORTH-based CWS [4].

F. The structure of the target compiler is described in ELMAGUIDE language or the compilers structure can be selected from the ones implemented in the ELMA-system.

The compilers generated with ELMA consist of the following subprocesses: lexical analysis, syntactical

analysis, transformation of a concrete syntax into an abstract one, semantic analysis, code generation.

Using these processes we can create one- and multipass compilers. Every pass has to be described in ELMAMETA languages. All these passes are connected into one compiling through ELMAGUIDE.

The exact description of ELMAMETA and ELMAGUIDE is given in [2,3].

ELMA as a structured programming support system for package writing

In ELMAGUIDE all descriptions and implementations are implemented using a technology, which is a development of Jackson's structured programming ideas. The most important principle is a strict separation of control and actions at any level of algorithm representation. Paraphrasing Wirth we can describe our ideas briefly.

algorithm = control + actions

The first part of the right side of the equality - control is built up by Jackson's structure and control conditions (for iterations and selections) which together determine the explicit behaviour of the computation process. The second part - actions make up the operational part of the algorithm. Every action in ELMAGUIDE language is a pointer to a module implemented in any programming language. Leaving low-level operations out of the language guarantees independence from the problem area.

The description style of the computing process can be called static, as it is fully defined before the process run. We have found it convenient to have some degrees of freedom in algorithm description and to allow for some refinements and modifications at runtime depending on intermediate results.

The dynamic aspect is implemented in ELMAGUIDE using system functions. This viewpoint is especially convenient for interactive work with an algorithm.

Example

As an example we demonstrate, how to describe and build ELMA systems monitor itself to create ELMA's interactive environment.

To work in this environment one has to use commands initializing certain processes or giving information, for instance, about feasible commands of the environment, i.e. commands menu. One can use the following processes: translator constructing, editing, debugging, executing a user's program, etc.

In our example we have a more detailed description for an algorithm corresponding to the creation of translators. This process is a dialogue between the ELMA system and the client. The latter has to define the parameters and the configuration of the translator. Even after taking this decision his wishes can easily be changed.

In the following we will give monitors Jackson schema (Fig. 1) using the standard structure of the translator from the dialogue to shorten the description and the client's work.

The representation of the monitor in ELMAGUIDE is given in Figure 2.

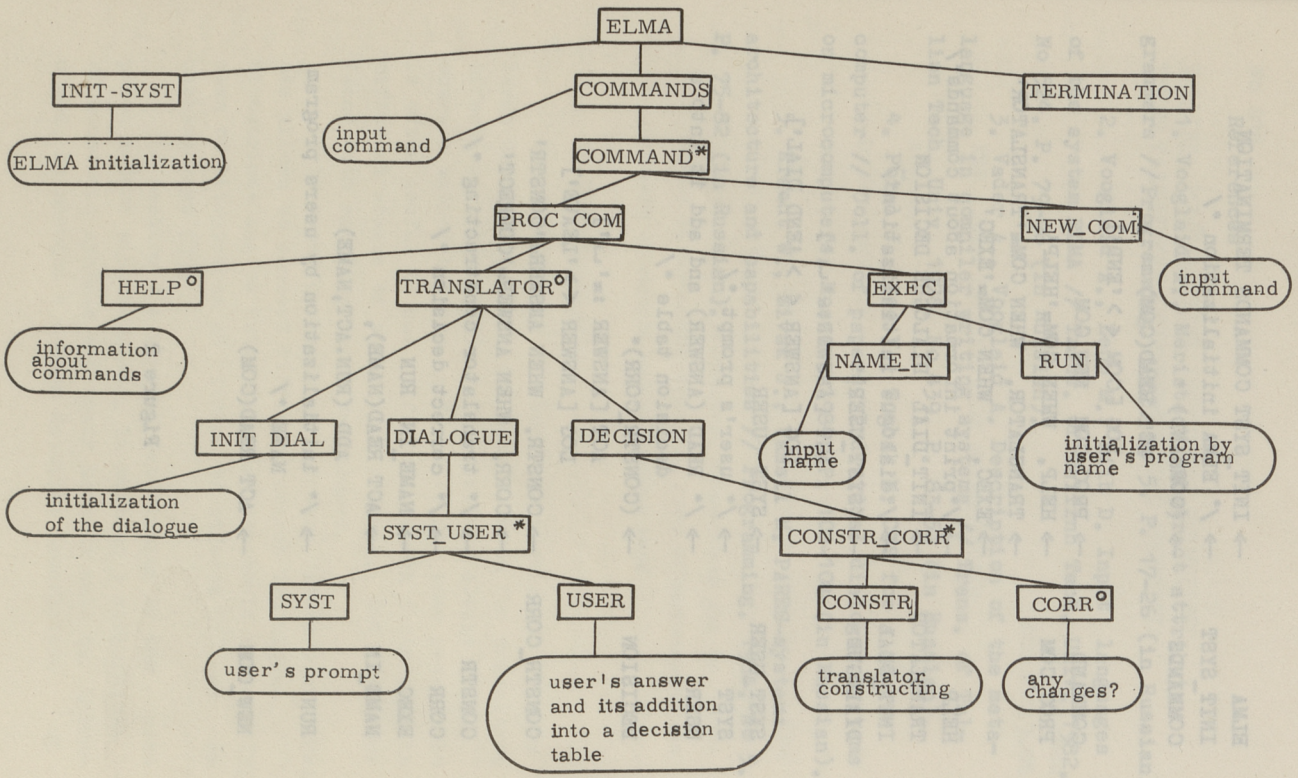

ELMA
INIT-SYST
COMMANDS
TERMINATION
ELMA initialization
input command
COMMAND*
PROC COM
NEW_COM
HELP°
TRANSLATOR°
EXEC
input command
information about commands
NAME_IN
RUN
INIT DIAL
DIALOGUE
DECISION
input name
initialization by user's program name
initialization of the dialogue
SYST_USER*
CONSTR_CORR*
SYST
USER
CONSTR
CORR°
user's prompt
user's answer and its addition into a decision table
translator constructing
any changes?

Figure 1

```
ELMA          -> INIT_SYST COMMANDS TERMINATION
INIT_SYST     -> /* ELMA initialization */
COMMANDS      -> (COMMAND) *
                     ACT READ(COM)
                     LOG [COM < > 'END']
COMMAND       -> PROC_COM  NEW_COM
PROC_COM      -> HELP.  WHEN COM='HELP'
              -> TRANSLATOR.  WHEN COM='TRANSLATOR'
              -> EXEC.    WHEN COM='EXEC'
HELP          -> /*print information about commands*/
TRANSLATOR    -> INIT_DIAL  DIALOGUE DECISION
INIT_DIAL     -> /*dialogue initialization*/
DIALOGUE      -> (SYST_USER)*
                     ACT [ANSWER:='␣']
                     LOG [ANSWER < > 'END_DIAL']
SYST_USER     -> SYST  USER
SYST          -> /* user's prompt */
USER          -> /* READ (ANSWER) and add it into a
                    decision table */
DECISION      -> (CONSTR_CORR)*
                     ACT [ANSWER :='␣']
                     LOG [ANSWER <>'LEAVE']
CONSTR_CORR   -> CONSTR.  WHEN ANSWER='CONSTR'
              -> CORR.  WHEN ANSWER='CORRECT'
CONSTR        -> /* translator constructing */
CORR          -> /* correct decision */
EXEC          -> NAME_IN  RUN
NAME_IN       -> ACT READ(NAME),
                     ADD (RUN.ACT,NAME)
RUN           -> /* initialization by users program
                    NAME */
NEW_COM       -> ACT READ(COM)
```

Figure 2

References

1. Vooglaid A., Meriste M. Abstract attribute grammars // Programming. 1982. No 5. P. 17-26 (in Russian).

2. Vooglaid A., Lepp M., Liib D. Input languages of the system ELMA // Trans. of Tallinn Tech. Univ. 1982. No 524. P. 79-96 (in Russian).

3. Vader A., Vooglaid A. Description of the meta-language in compiler writing systems // Trans. of Tallinn Tech. Univ. 1978. No 439. P. 83-52 (in Russian).

4. Pöial J., Soo V., Tombak M. CWS for a micro-computer // Coll. of papers. Individual dialogue systems on microcomputers. Moscow. 1984. P. 102-104 (in Russian).

5. Kracht W., Eivak J., Vassil M. PARES-system: architecture and capabilities // Programming. 1984. No 5. P. 75-82 (in Russian).

No 614

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED
ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

P. Vyhandu
UDK 681.3.06

NEW IDEAS IN DATA BASE SEGMENTATION

Abstract

In this approach a relational data base is designed such that the data more often occurring in the same query belongs to one relation. In this way the need for creating access paths is reduced. Bitmatrices allow to connect different search techniques. Bitmatrices can be interpreted as monotonic systems, rows of bitmatrices as bitmaps of objects and as hashing keys.

1. Introduction

The fast search has become a problem in very large data sets. There are many methods for organizing effective search [1,2], but situations always arise where a method becomes ineffective for some reason. It is often useful to use different search techniques in different stages of the search process. But we are faced with matching those techniques.

This paper presents an approach, where bitmatrices use allows to connect different search techniques. Bitmatrices can be interpreted as monotonic systems [3], rows of bitmatrices as bitmaps of objects [6] and as hashing-keys [7].

In section 2 a general description of the proposed method will be given. In section 3 definitions of monotonic systems and their kernels will be introduced. In section 4 a more detailed description of

data base design and search process will be given.

2. Methods for quick search

In relational data bases data is maintained in relations, where each relation contains one record type. The record type usually corresponds to a real world concept. In our approach relational data base is designed such that the data more often occurring in the same query belongs into one relation. In this way the need for creating access paths, which is a time-consuming process, is reduced.

To decide which data must be maintained in one relation, an analysis of the data set is made first. After forming relations directories are built for them. Records in the relation are grouped to have similar records in one group. On these groups hierarchical structures, called directories, are built. Using directories enables us to determine the groups which cannot contain records satisfying the given query in the early stage of search.

In such a data base a thorough search on a small part of data base is implemented and search time becomes much shorter.

Concepts of the monotonic system and its kernel are used for the preceding analysis of the data set and clustering. Bitmatrices serve as so-called upper structures on which the main search is performed.

The structure of the data base depends completely on data characteristics and it is impossible to predict the number of relations in the data base, that of groups in a relation and that of records in one group. For that reason the extendible hash method fits well to find the real records. This hash method has properties which make it very convenient to use when the number of records in the hash table is not known and the hash keys are in the binary form. The hash method is not dealt with here, but its description can be found in [4,7].

3. Monotonic systems. Kernel

As it was observed in the previous section data ordering must precede the design of the data base. Concepts of monotonic system and its kernel are used for that [3]. First definitions and kernel splitting algorithms are given. In case of data set analysis and record grouping we have to interpret data matrices as different monotonic systems. Therefore different algorithms for the kernel splitting are introduced.

3.1. Definitions

Let us suppose that there is a system W with a finite number of elements. Each element has a numerical measure of its weight (influence) in the system. Let us suppose further that for every element $\alpha \in W$ there is a feasible discrete operation which changes the weight of α as well as the weights of any other element β of the system. If the elements on W are independent, then it is natural to suppose that a change in the weight of α does not change the value of any element β. System W is called monotonic, if the operation of the weight change of any element $\alpha \in W$ brings about changes on the weight levels of other elements only in the direction in which α itself is changed.

To use the method of monotonic systems we have to meet two conditions:

1) There has to be a function π which gives a measure (weight) $\pi(W)$ of influence for every element W of the monotonic system W.

2) There have to be rules f to recalculate the influences of the elements of the system in case there occurs a change in the weight of one element.

A kernel of a monotonic system W is defined as a subset H of its element on which the global minimum of function

$$F(H) = \max_{W \in H} \pi(n,H)$$

is reached.

3.2. Kernel splitting (one-dimensional case)

Let us have a data matrix $A=(a_{ij})$; $i=1,\ldots,M$; $j=1,\ldots,N$. Let us interpret matrix A as a monotonic system, the elements of which are the rows a_i of A. Kernel is a subset H of its elements on which the global minimum of function

$$F(H) = \max_{a_i \in H} \pi(a_i, H)$$

is reached.

To measure the influence of an object on the system, we define the function

$$S_i = \sum_j (2n_{ij}^2 + 3n_{ij} + 1), \qquad (1)$$

where n_{ij} is the frequency of value a_{ij} in the histogram of the j-th attribute.

To split the kernel of the monotonic system we will find sequentially the elements with the greatest influence and add copies of them to the system. It has been made clear that addition of an element to the system alters the influences of all other elements.

Adding an element K to the system changes the influence for any other element i for

$$S(n_{ij}+1)-S(n_{ij}) = \sum_j \delta[2(n_{ij}+1)^2+3(n_{ij}+1)+$$

$$+1-2n_{ij}^2-3n_{ij}-1] = \sum_j (4n_{ij}+5), \qquad (2)$$

where $\delta = \begin{cases} 1, & \text{if } a_{kj} = a_{ij}, \\ 0, & \text{if } a_{kj} \neq a_{ij}, \end{cases}$

i.e. we sum the frequencies only for these attributes of an object i, when the value of an attribute matches the value of an attribute of the object k. If objects k and i have no matching attribute values, addition of the element k to the system does not influence the element i.

The kernel splitting is performed by ALGORITHM 1. In this algorithm elements are not actually added to

the system, only the influences are computed accordingly. By an added element we mean a labeled element with the greatest influence.

ALGORITHM 1

Step 1. For each element a_i, i=1,...,M compute

$$S_i = \sum_j (2n_{ij}^2 + 3n_{ij} + 1).$$

Step 2. Find max S_i, memorize k=i, S_k.

Step 3. For each element compute $S_i = S_i + P_i$, where $P_i = 4T+5L$, and T is the sum of the frequencies of the matching attribute values of elements k and i, and L is the number of matches.

Step 4. Find max S_i for all elements not added to the system.

Step 5. If max $S_i < S_k$, then end of the algorithm.

Step 6. Memorize k=i, S_k.

Step 7. In the auxiliary table of frequencies for every value for the element k add 1.

Step 8. Go to step 3.

On step 5 reaching the extremum of the function π is controlled. All the elements memorized before reaching the extremum of function π , belong to the kernel of the monotonic system.

3.3. Kernel splitting (two-dimensional case)

Let us have a data matrix $A=(a_{ij})$, i=1,...,M, j=1,...,N. Let us interpret matrix A as a monotonic system, the elements of which are rows a_i and columns a_j, a_i, $a_j \in A$. The kernel is a subset H of its elements on which the global minimum of function

$$F(H) = \max_{\substack{a_i \in H \\ a_j \in H}} \pi (a_i, a_j, H)$$

is reached.

Let us have for every element a_i a number of its incidences n_i. Then the number of incidences having the j-th attribute is equal to

$$h_j = \sum_i n_i \delta , \tag{3}$$

where $\delta = \begin{cases} 1, & \text{if } a_{ij} = 1, \\ 0, & \text{if } a_{ij} = 0. \end{cases}$

Initial influences can be computed as follows

$$g_i = n_i \sum_j (2h_j^2 + 3h_j + 1)\delta \tag{4}$$

for the elements a_i and

$$g_j = h_j(2h_j^2 + 3h_j + 1) \tag{5}$$

for the elements a_j.

To split the kernel we will add the elements with the greatest influence to the system. Adding an element a_j to the system, the influence of every other element a_r, $r = 1,\ldots,M$ grows by

$$d_r = n_r(2h_j^2 + 3h_j + 1)\delta . \tag{6}$$

Adding an element a_i to the system, influences of elements a_j can be computed by a formula

$$g_j = k(2k^2 + 3k + 1),$$

where

$$k = h_j + n_i \tag{7}$$

and the influence of the element a_r, $r=1,\ldots,M$ grows by

$$\begin{aligned} \Delta r &= [(2k^2+3k+1)-(2h_j^2+3h_j+1)]\, n_r = \\ &= n_r[2(h_j+n_i)^2+3(h_j+n_i)+1-2h_j^2-3h_j-1] = \\ &= n_r(4h_jn_i+2n_i^2-3n_i) = n_rn_i[2(n_i+2h_j)-3]. \end{aligned} \tag{8}$$

To add elements to the monotonic system up to the kernel splitting, we use ALGORITHM 2.

ALGORITHM 2

Step 1. For each j, j=1,...,N compute the number of incidences h_j by formula (3).

Step 2. By formula (4) and (5) compute initial influences g_i and g_j.

Step 3. Find the element with the greatest influence, i.e

$$P=\max_{g \in g_i \cup g_j} g$$

Step 4. Memorize $k = \begin{cases} i, & \text{if } g \in g_i \\ j, & \text{if } g \in g_j \end{cases}$ and $S_k=P$.

Step 5. If $\max g \in g_i$, go to step 8.

Step 6. For each r, r=1,...,M compute d_r by formula (6) and add it to the influence g_r.

Step 7. Go to step 10.

Step 8. For each j, j=1,...,N compute influences g_j by formula (7).

Step 9. For each r, r=1,...,M compute $g_r=g_r+\Delta r$, where Δr is computed by formula (8).

Step 10. Find $p=\max_{g \in g_i \cup g_j} g$. If $S_k < P$ go to step 4.

Step 11. End of algorithm.

4. Data base design

Data base design is divided into two stages: analysis of the data set for partitioning data items into relations and build-up of directories.

4.1. Data set analysis

For a relational data base we have the following definitions [5].

Attributes are indentifiers taken from a finite set $A_1, A_2, \ldots, A_n$. Each attribute A_i is associated with its domain, denoted by $DOM(A_i)$, which is a set of possible values for that attribute. We use the

letters A, B,..., for single attributes and the letters X,Y,..., for sets of attributes.

A relation on the set of attributes $\{A_1,A_2,...,A_n\}$ is a subset of the Cartesian product $DOM(A_1)xDOM(A_2)x...$ $...xDOM(A_n)$. The elements of the relation are called tuples. A relation R on $\{A_1,A_2,...,A_n\}$ is denoted as $R(A_1, A_2,...,A_n)$.

Relational algebra as a data manipulation language is introduced. There are two basic operations of interest for us: projection and natural join.

The projection of a relation R(X,Y,Z) over the attributes in X will be denoted R[X], and defined by $R[X]=\{x \mid \exists y \exists z : (x,y,z) R\}$.

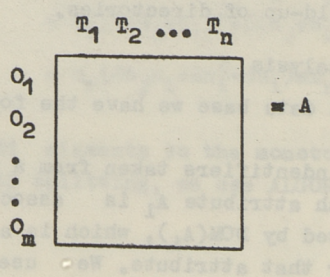
The natural join operation is used to make a connection between attributes that appear in different relations. Let R(x,y) and S(x,z) be two relations; then the natural join R*S is the set of $\{(x,y,z) \mid (x,y) \in R$ and $(x,z) \in S\}$; R*S is a relation defined over the attributes {X,Y,Z}.

Let R(X,Y,Z) be a relation; we shall say that R is decomposable if there exist two relations S and T, such as:

a) S and T are the projections of R: S=R[x,y], T=R[x,y]

b) the natural join of S and T is R: R=S*T.

Using the natural join operation all the relations in the data base can be connected into one relation U. A model matrix A can be put into correspondence with the relation U



where $\Omega = \{O_i\}$, $i=1,\ldots,m$ is the set of object types,

$\tau = \{T_j\}$, $j=1,\ldots,n$ is the set of attributes

and

$$a_{ij} = \begin{cases} 1, & \text{if an object type } O_i \text{ has an attribute } T_j, \\ 0, & \text{otherwise.} \end{cases}$$

For each object type the number of its incidences is also given.

Such a model can be interpreted as a monotonic system described in section 3.3. Using ALGORITHM 2 all kernels of the system are separated.

Let as suppose that the number of separated kernels is equal to p. The kernel is denoted by K_s, $s=1,\ldots,p$. Using the projection operation, relation U can be decomposed in the following way:

a) $R_1 = U[X_1]$, $R_2 = U[X_2],\ldots,R_p = U[X_p]$,

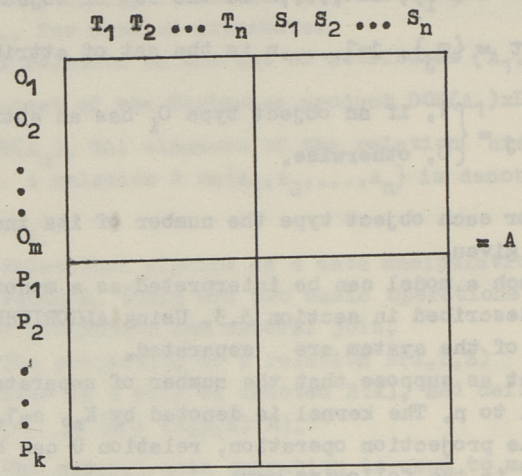
b) $U = R_1 * R_2 * \ldots * R_p$

where $X_s = \{a_j\}$, $a_j \in K_s$ and tuples of relation R_i are incidences of the object types corresponding to the elements $a_i \in K_s$.

The relations created contain similar data items.

But it were more effective, if we could store data that occurs in the same query in one relation. Then the need to use operations of relational algebra is reduced and therefore search time decreases.

To achieve that, queries and connections determined by the queries must be considered on the model matrix A in addition to object types and attributes. The model matrix A is presented in the following form:



where $\Omega = \{O_i\}$, i=1,...,m denotes the set of object types,

$\tau = \{T_j\}$, j=1,...,n denotes the set of attributes,

$\pi = \{P_i\}$, i=1,...,k denotes the set of queries,

$\Sigma = \{S_j\}$, j=1,...,l denotes the set of connections,

and

$$a_{ij} = \begin{cases} 1, & \text{if an object type } O_i \text{ has an attribute } T_j, \\ 0, & \text{otherwise, } i=1,\ldots,m,\ j=1,\ldots n; \end{cases}$$

$$a_{ij} = \begin{cases} 1, & \text{if a query } p_i \text{ contains an attribute } T_j, \\ 0, & \text{otherwise, } i=m+1,\ldots,m+k,\ j=n+1,\ldots,\ n; \end{cases}$$

$$a_{ij} = \begin{cases} 1, & \text{if a query } P_i \text{ determinss a connection } S_j, \\ 0, & \text{otherwise, } i=m+1,\ldots,m+k,\ j=n+1,\ldots,n+1; \end{cases}$$

$$a_{ij} = \begin{cases} 1, & \text{if a connection } S_j \text{ contains an object type } O_i, \\ 0, & \text{otherwise, } i=1,\ldots,m,\ j=n+1,\ldots,n+1. \end{cases}$$

For each object type the number of its instances and for each query the frequency of occurrence is also given. Analogously to the preceding model, matrix A is interpreted as a monotonic system, all the kernels are separated and new relations are created. Record types

in relations are structures where data items are not only similar to each, but are closely connected through occurrence in the same queries.

4.2. Bitmaps and directories

Let us have in one relation m records and n attributes, where the j-th attribute (j=1,...,n) has N_j different values. Then a bitvector of length $d = \sum_{j=1}^{n} N_j$ can be created for each record.

The elements of the bitvector are filled as follows:

$$a_j^k = \begin{cases} 1, & \text{if record has value } k, \\ 0, & \text{otherwise, } j=1,\ldots,N_j. \end{cases}$$

This bitvector is called a bitmap. In [6] the approach of directories built of the bitmaps is given.

Let us interpret the set of bitmaps as a monotonic system. Using ALGORITHM 1 from section 3.2 all kernels are separated. Using disjunction operator one superbitmap is formed of bitmaps belonging to one kernel. This superbitmap is called an address.

Out of the formed addresses a new monotonic system can be made and the process can be repeated. Using that process recursively a hierarchical structure, called directory, is formed.

4.3. Search process

If the data base is created using the methods described in this paper, a relation and its directory will be in the form shown in Figure 1.

The data base consists of relations. A set of bitmaps corresponds to each relation on which a directory is built. Records in relations are connected with their bitmaps via extendible hashing. Bitmaps and directories are called upper structures of the data base.

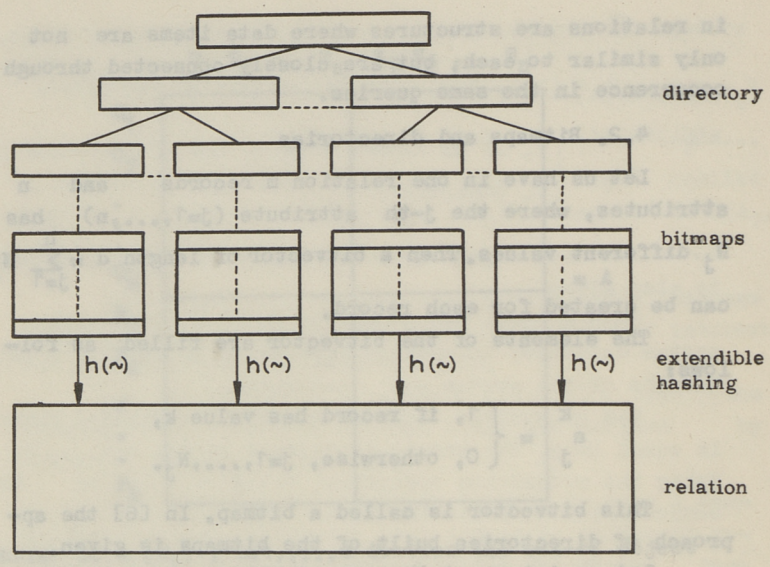


Figure 1

The search process runs as follows:

According to a given query we have first to determine in which relations the needed data is stored. After that directories of these relations are examined. If we find out that the group the given address is representing, cannot contain the needed data, we do not investigate it any further. A sequential search is performed on the set of bitmaps which may meet query conditions. Using extendible hashing the needed records are quickly located.

The search time is reduced because

(I) of a great probability that the needed data is stored in one relation. The need for relational algebra operations is decreased.

(2) even in the realm of one relation the sieving process cuts off the number of objects on which the full search is performed.

References

1. Martin J. Computer data-base organization. Prentice-Hall, 1977.

2. Salton G. Dynamic information and library processing. Prentice-Hall, 1975.

3. Mullat J., Vyhandu L. Monotonic systems in scene analysis // Symposium. Mathematical Processing of Cartographic Data. Tallinn, 1979. P. 63-66.

4. Fagin R., Nievergelt J., Pippenger N., Strong H.R. Extendible hashing, a fast access method for dynamic files // ACM TODS, 1979. Vol. 4. No 3. P. 315-344.

5. Delobel C., Pichat E. The design of relational information system according to different kinds of dependencies // COMPSAC 78. Proc. IEEE. Chicago, 78.

6. Выханду Л.К., Выханду П.Л. Быстрый поиск на битматрицах // Тр. Таллинск. политех. ин-та. 1983. № 554. С. 49-60.

7. Выханду Л.К., Выханду П.Л. Синтез метода адресных книг и расширяющегося хэширования // Тр. Таллинск. политех. ин-та. 1984. № 568.

No 614

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED
ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

J. Laast-Laas
UDK 681.3.016

DATA BASE INFOLOGICAL DESIGN IN PRACTICE

Abstract

Some general standpoints of the data base infological approach are described. Practical data base design problems related to the examination of user's needs, data base decomposition and logical integration of data are dealt with.

By the infological approach we mean a special and useful way of handling ideas related to data base design. It enables us to use a wide variety of system analysis, data structure design, statistical and other methods. We have used the approach in management information system (MIS) data base design since the mid 1970s.

In this paper we concentrate on some general data planning and design problems, solved using the infological approach. First of all some general standpoints will be presented.

In an organization/enterprise a distinction can be made between at least two levels of activity: primary actions (e.g., making shoes in a shoe factory or repairing cars in a car service) and secondary actions (e.g., book-keeping, statistical reporting, etc.). All the primary actions along with the used and produced resources are treated as the object-system. Secondary actions with their resources (incl. information) make up a monitoring system.

The object-system is a functioning piece of reality being managed or sometimes only monitored. In any par-

ticular case it can be defined just as detailed and as wide as necessary. As there exist several different aspects of monitoring the same object system, several respective monitoring systems could be determined.

A data base must be designed as a model of the object-system's essential behaviour.

A data base designer should stay as close to the object-system as possible and as far away from the existing data processing system as possible. There are three arguments for that:

1) the object-system is common for all possible monitoring aspects
2) the object-system is simpler to be presented in a data base than the monitoring system
3) the architecture and behaviour of the object-system are more stable than monitoring and data processing problems.

Object-system modelling is natural in theory and simple in selected school examples, but in practice considerable difficulties are encountered. The object-system is an abstract piece of reality, the stretch and configuration of which are rather vague. There are no ready concepts for its determination and description. To determine what belongs to the object-system, what is significant in it and how detailed it should be modelled, the existing information system and needs of prospective users have to be examined anyway.

The infological approach enables MIS designers and users to find a common language between each other.

Architects, for example, seem to be in a much better situation than MIS designers, as their clients know quite exactly what they want and usually present clear functional and technical requisitions. A MIS designer gets his task in a very vague form. The "requisites" often turn out to be occasional examples of queries. The very first question that a MIS designer has

to answer is: What should it be that the client would need? It seems that a designer has to know more about the MIS prospective user's needs and problems than they know themselves. Apart from that he must be able to predict their future needs.

If the infological approach is chosen, the situation is not so contradictory and hopeless. Interviewing managers or examining various documents, a designer-infologist is actually concentrating only on the object-system. He can ignore almost everything he is told about data processing difficulties, management problems, etc.

> Record these conversations to go them through in detail later. Then you will definitely be able to understand and use that information better.

An object-system can always be described in simple and commonly understandable terms. On these counts the common language between MIS designers and managers can be found quickly and surprisingly well.

> First of all learn what a manager needs information about, do not try to make clear what information he needs, how the data should be processed and presented.

The infological approach avoids "transplantation"

A new information system is never built up on an empty place. There exists an old one, which obviously is not good enough, but still has its deep roots in people's minds and bureaucracy. Some elements of old information system which have lost their initial meaning and significance in a new environment are likely to appear as useless relics and may cause much harm. Paper documents, for example, are equivalently transformed into file records, manual document-processing-routines are replaced by the corresponding computer programs. Stiffness, redundancy and complexity are obvious in such a data processing system.

Usually "transplantation" does not take place in such

a drastic way. More often some design is performed to make data organization more computer-like and less redundant, but the "transplantation" idea remains. A set of information processing tasks usually called a "functional subsystem", but actually being a set of routines from the old system, is treated as a separate MIS block. Difficulties with matching blocks arise later. Reconstruction of some particular blocks involves corrections in others... The MIS can never be accomplished.

What is actually wrong? - A logical mistake has been made at the start of the design. The information system was physically split by information processing tasks. The tasks are not invariant and have dynamic complex relationships between one another. Thus the MIS architecture becomes complex and dynamic as well. It will need frequent restructuring.

> Do not start with examination or assignment of any data processing or monitoring tasks! Try to determine the object-system first of all, study its behaviour, draw the framework of its model! Decompose the model reasonably!

The infological approach provides principles for data base decomposition

We distinguish between physical and logical decomposition. In both cases every separated data base remains a model of something from the object-system. Physical decomposition corresponds to the object-system configuration dismemberment. A data base of an organization, for example, can be split into local data bases for each local branch. Notice that it means reduction of physical extent of data in single data bases, but their logical complexity remains almost the same as it were the general data base. Not to lose an overall view, either an additional general data base or a general view handler must be provided.

Logical decomposition means splitting a data base

schema into subschemas. Each separate data base has a considerably simpler logical structure than the general one. It will be a model of a logical piece of the object-system (e.g., "inventory", "personnel", "manufacturing"). Naturally, also the schemas of logically decomposed data bases will overlap and it involves some redundancy and updating problems. Still it is necessary for general applications. Logical decomposition is quite complicated and needs effective methods to be accomplished properly.

Logical and physical decompositon can also be used in various combinations.

┌──────────────────────────────┐
│ Use logical decompositon if possible! │
└──────────────────────────────┘

The infological approach reduces integration problems

The new MIS is supposed to become a monitoring system or a part of it. The aim is to integrate different monitoring systems for more effective data processing and for provision of additional monitoring facilities.

The existing monitoring systems are connected to one another, but they are functioning separately, they have their own internal data items, structures and processing technologies, which are often incompatible with one another. Attempts to match them mechanically yield a "hodge-podge". There is also no sense in seeking general concepts for specifying different monitors to achieve the integrity by generalization. It means building up a supersystem above the existing monitors. The following arguments oppose that approach:

- The monitoring systems are complicated and abstract enough; more general specification needs further abstraction, involves new terminology and considerably enhances the complexity of the entirety.

- Being carried by trends of management and economical policy, the monitoring systems are in continuous development. It is impossible to predict potential changes. One cannot be sure that some of such changes

will not destroy the whole MIS architecture some day.

- The major aim of integration is not monitoring monitors but replacement of old monitors or assistance of monitoring.

The use of the infological approach helps avoid logical difficulties in data processing integration. A good infological model meets the needs of different monitoring systems, remaining comparatively independent of changes in user's information needs and external data structures. Top-down general design and reasonable data base decomposition provide compatibility between separate data bases. Bottom-up detail design and MIS realisation assure urgent management data processing goals to be achieved first of all.

No 614

TALLINNA POLÜTEHNILISE INSTITUUDI TOIMETISED
ТРУДЫ ТАЛЛИНСКОГО ПОЛИТЕХНИЧЕСКОГО ИНСТИТУТА

A. Renzer
UDK 681.3

INFORMATIONAL ASPECTS OF BUSINESS ANALYSIS SYSTEMS DESIGN

Abstract

The paper reviews the universal principles of business information systems design, considering the intrinsic features of business information and the ways of its automatic processing.

1. Introduction

Improvement of the existing management system, including the business analysis function, is to be achieved by introducing automatic information systems either as certain subsystems of an automatic management system (AMS of an enterprise, or as relatively self-dependent systems.

The ultimate result of introducing an automatic business analysis system (ABAS) greatly depends on how systematic and complex the analysis method is, and how veraciously the actual processes are reflected by the system of indexes used. So far no adequate methods for automatic analysis of business enterprise functioning or design of interactive business information systems have been offered.

The development of computer technology is likely to have great impact on the structure of systems above: e.g. system design is affected by the emergence of home-made micros. Till the end of the 1980s, such systems are to be designed consisting of two levels - the first one including a dozen micros for data acquisition and preliminary processing, with the second level supported by

a medium level computer for advanced processing. With the help of the latter all data is compressed into one logical unit under the supervision of a data base management system to assure the accessibility of all data to every user. Below an example of a system with such architecture will be shown.

The first section of the paper is dedicated to the intrinsic character of business information and the ways of its automatic processing. In the second section the role and specification of business analysis in the management system will be reviewed. In the third section we will demonstrate an example of the ABAS architecture and bring out some issues of its information funds.

2. Business information and the ways of its automatic processing

Business information is a set of data that reflects the economic and social activities of a business enterprise via a system of natural, financial and other indexes. Hence, the bulk of business information is digital in essence, and its automatic processing with various mathematical and other methods is feasible. Through a closer examination of data processing problems tremendous difficulties are likely to arise due to the nature of business information.

First, business information emerges, is transmitted, processed and used at different levels, in different hierarchical substructures of management, and even at one level - by various management subsystems. Frequently this causes double acquisition of information, genesis of indexes having the same economic origin and essence, but different numeric values.

Second, business information is not homogeneous in view of time characteristics. Parts of it are relatively permanent (norms, long-range prognoses, etc.) but a lot of data is acquired which is changing during the technological process. The cycles of formation and periods of processing evaluation of different business informa-

tion flows do not coincide. All this influences the modelling of informational processes, and without the use of mathematical models automatic processing of business information would be unthinkable.

Third. A bulk of diverse analytical information, the contents and economic interpretation of which are not always trivial is obtained from the initial business data by various economic and mathematical methods. Thus, authenticity and representativeness of analytical information are the issues still under discussion.

We are faced with certain extra difficulties in the area of so-called information needs, e.g. when seeking the answer to the questions of whom, when and how much of the acquired information is to be conveyed. It is common knowledge that an outsize flood of information can do as much damage to management as the absence of some vital figure in the moment it is needed. Anyone can see the value of managing information received only after decision making time has elapsed. Therefore it is imperative that the information flow be organized so that every managing action is justified on sound grounds and well timed by supportive business data [2, p. 20]. Under other equal conditions better management is achieved by those who have obtained more thorough qualitative information about the social and economic processes in their business enterprise.

The quality of information is determined by its:

(I) fullness, truthfulness and accuracy

(2) timing of data acquisition and strict periodicity of its passing

(3) ability to unveil negative phenomena in the social and economic processes in the enterprise.

In terms of its contents business information can be classified as consisting of:

(I) data of the final products, their composition anda quality

(2) data of production methods

(3) final product quantity - actual and calculated

(4) scheduled and actual production expenditures

(5) information of non-accounting nature.

There are several classifications of business information based on its stability, ways of obtaining, nature of contents, applications, etc. But in any of those classifications certain specific features of business information, impeding its processing, and automatic processing, in particular, are to be perceived. These are the various levels of information formation, transmission and use, the incomparability, interwovenness and independence of indexes; as well as the doubling and diversity of data, non-corresponding cycles of formation and processing, etc.

Therefore it is indispensable to create accurate methods for business information systems design [I, p.18], including methods of organizing information support for an ABAS.

The importance of methods and methodology issues is accounted for by the character of business information. It is common knowledge that every business enterprise functions in a specific and unique environment. Hence there exists the singularity of managing every enterprise as a whole and on every hierarchical level separately taken. No uniform or pattern-created data processing system can meet the needs of managing specific enterprise, including accounting, planning and analysis automation. The creation of individual systems must proceed from large instrumental means which minimize additional software writing and assure tunability of the system to a particular enterprise by means of adaptation.

Some attempts in the line of creating ABASs have been made by a research team at Tallinn Technical University. The future systems must feature the following attributes and capabilities:

(I) flexibility - the systems must be tunable to a multitude of different enterprises

(2) orientation to end user (even on the cost of additional implementation expenditures), including:

- no programming on end user level

- the dialogue with the computer in a language close to the end user's professional one

- highly developed non-numerical information output (colour graphic resources, sound and speech outputs, etc.)

- run-time instructions and menus of possible operator actions offering capability

(3) the system must be supported by modern computer hardware (micros and minis linked to medium computers, teleprocessing systems, automatic data collection devices, etc.)

(4) maximum use of the existing means, involving instrumental systems and modern data base systems.

Only if these demands are satisfied, the systems can be designed and implemented within acceptable time limits and the transition to automatic management will be painless.

3. The role of automatic business analysis in enterprise management

In management automation an undesirable approach is often met: a multitude of AMS subsystems are designed corresponding to various attributes of grouping the involved tasks - management functions, time characteristics, administrative structures, and the like. These would be relatively independent subsystems (and frequently with incompatible software) which are not supported by an integral information fund. Apart from that, the designers are carried away by planning subsystems on administrative basis (systems of units or subjects of management) or by production types (main, subsidiary, production preparation etc.), forgetting all about the functional characteristics. In the outcome, some of the business analysis issues are dealt with in good many different management subsystems, but some are left quite untouched. Most often the operative management is automatic, but prognosticating, business development evaluation, etc. stand out of the reach of the automatic information system.

Many authors, including those of [1], [2] and [5] emphasize the necessity for separation of a relatively

detached self-dependent business analysis system. This view derives from weighty arguments that we will now proceed with.

I. Business analysis includes the whole system of production and economic activities in all types of management (operative, current and perspective), it carries out all kinds of analytical tasks related to all management subjects on all the existing levels.

2. Business analysis is an inseparable part of such economic procedures as planning, control and accounting. To handle the issues of analysis under other management subsystems appears to be insufficient, because the analysis will not be systematic and complex. The approach would be one-sided due to the specific features of the other subsystems, because they are designed for altogether different purposes, and so many influential factors would not be taken into consideration when studying the economic processes.

3. The mission of business analysis is highly specific and its tasks are of somewhat independent nature. They include:

(a) evalution of the enterprise condition and determination of the development trends in it

(b) evalution of the seriousness of deviations from certain normative or scheduled figures

(c) investigation of the causes for deviations and influences of different factors

(d) establishing material and productive resources.

4. The introduction of automatic business analysis systems is to be considered inevitable for management decisions support in prognosticating, planning, auditing and regulating of all of management subjects.

Like other functions of management, business analysis cannot have identical contents on all the hierarchical levels; different enterprise functions (organization of production and labour, techniques and technology, economics and social development) must be analysed in different ways. Specific pieces of information about the enterprise functioning have to be woven into an

integral cloth of data for one reason only - to obtain a complete, thorough and comprehensive picture of the economic processes in that particular business enterprise. Under the supervision of prof. A. Sheremet ([6] et al.) sufficient methods for the complex business analysis have been developed, but no implementations of real automated systems have followed yet.

Thus, we have discussed the necessity for introducing an automated business analysis system. What should such a system look like? In technical terms it has to be a computing set, usually comprising the following components:

1. Hardware (main and auxiliary computers, means of data acquisition and transmission, terminal stations, etc.). As it was suggested in papers [3] and [4] , it is useful to organize the data collection and preliminary processing on microcomputers (lower level) with advanced processing supported by a medium computer (higher level).

The higher level is usually a part of a larger system (integrated MIS or other) and thus has a more complex structure.

2. Software (one possible solution is given in [4]).

3. Infoware which means the technology of collecting, storing and passing information as added to the body of information itself.

4. Maintenance personnel - its amount and hierarchy depend on the scale of economic activities.

5. Technical documentation for maintenance personnel.

6. Technical documentation for users, including mainly manuals and instructions for different strata of users. Commonly the lower level systems are exploited on two levels - first, the end users (management personnel, technical staff, and, second, the so-called system administrator, who has deeper knowledge of the system and who directs its usage.

7. Administrative and legal documents that master the relationship between users and the computing set.

Special attention should be paid to the informational interface between different administrative units (an issue of infoware) and to the questions of hardware as far as the automatic systems design is concerned.

4. Architecture and infoware of automatic business analysis systems

One of the keystones of any information system is the organization of information supplies and storing. The following goals are to be achieved:

(1) determine the structure, sources and methods of acquisition of the initial data

(2) determine data set contents, amount, logical organization and terms of data preservation

(3) determine the output information amount, periodicity and ways of transmission

(4) grant the accessibility of all information to any qualified user.

The information system must guarantee the provision of various normative, scheduled, prognostic, accounting and other figures for calculating permanent reports as well as for meeting every unexpected need in that information. For every business enterprise an individual system of economic and social indexes must be developed to cover all its unique features, but a possibility for comparison with other enterprises must be assured.

The question of recording the whole initial information must be taken care of in accordance with the valid legal regulations and requirements of the particular enterprise. Storing the intermediate computing results cannot be justified unless they are included in certain time-rows or other useful figures that are often referred to. The value of other intermediate results, if need be, can be easily established by any reasonable system of occasional data base retrievals.

The issues of information recording are closely

linked to the methods of its acquisition. As it was suggested in [3], we should consider organizing the ABAS consisting of two subsystems, where data collection and preliminary processing are carried out by the lower level system supported by microcomputers located in various administrative units. The acquired data (but not all of it) is sent to the main analysis system of the enterprise, where advanced statistical and other analyses are made with methods demanding more computer resources. Of course, the appearance of sufficiently powerful micros that can be physically and logically joined into networks, would allow the implementation of ABASs without the present day medium and large computers use. All data sets on individual external storage units (e.g. different files of one logical data set) would be obtainable by all users, more precisely, by the whole management personnel, who has access to the computers.

Still even the mixed usage of micro- and medium computers would solve good many problems of the information support: the micros in every hierarchical substructure hold a complete set of initial information (it is mainly the primary documentation) about that particular production unit. The local management has complete access to that information via micros and can use it freely. When a need should arise for information about other production units, it may easily be retrieved from the main data base through the same information channels by which their own data is sent to the IDB. So the frequency of the main base calls is reduced and the IDB can be organized on a much less powerful (and cheaper) computer.

A business information data base should comprise the following main blocks:

(1) operative, book-keeping and statistical accounting information

(2) operative, current, perspective and long-range scheduled and normative figures

(3) primary documentation (at least to some extent)

(4) information of non-accounting nature (board meetings' documentation, instructions, memos, etc.).

The exact composition of information is varying from one enterprise to another and so no ready-made lists can be offered. The rising interest towards the development of logical data base design methods should be applauded.

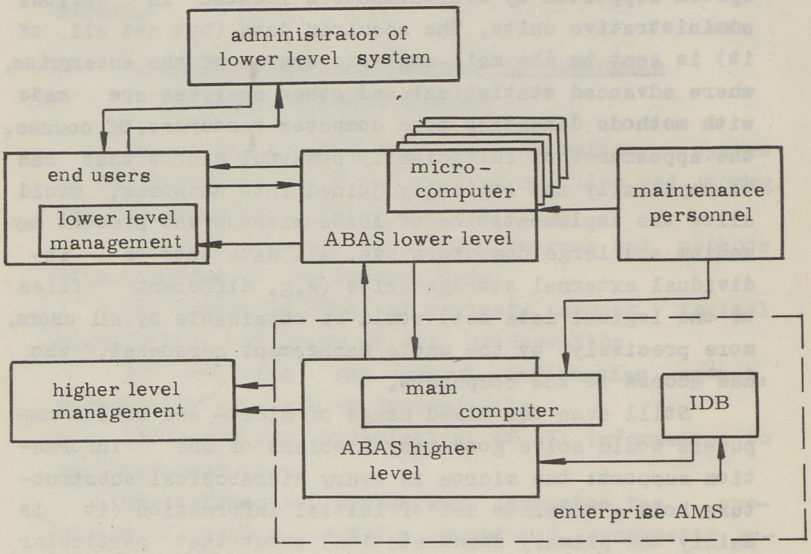


Fig. 1. Possible architecture of an ABAS

In conclusion, we should note that much obscurity is still present in the design of automatic business analysis systems which are relatively independent and self-supporting, use the primary documents of the enterprise as source information, and support all management levels and administrative substructures with appropriate analytical data. A sharp need is felt for exact design methods of the automatic analysis procedure and the corresponding information systems alongside the necessity for a methodology development of information support for business analysis - the state of infoware strongly influences the eventual result of the system introduction.

Here we have outlined some issues of automatic business information processing and automatic business analysis systems to determine the shortest ways to feasible creation of such systems.

References

I. Аксененко А.Ф., Пантелеев В.А. Информационное обеспечение комплексного экономического анализа в АСУП // Бухгалтерский учет. 1984. № 2. С. 17-20.

2. Либерман В.Б. Автоматизация расчетов себестоимости продукции на машиностроительных предприятиях. М., 1982.

3. Рензер А.В. Система автоматизированного анализа экономической деятельности предприятия // Тр. Таллинск. полит. ин-та. 1983. № 554. С. 121-127.

4. Рензер А.В. Система автоматизированного экономического анализа на базе ПОК "АРМ-Экономика" // Тр. Таллинск. полит. ин-та. 1984.

5. Самборский В.И., Луцак И.И. Экономический анализ на машиностроительном заводе. Киев, 1981.

6. Экономико-математические методы в анализе хозяйственной деятельности предприятий и объединений. М., 1982.

ТАЛЛИНСКИЙ ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Труды ТПИ № 614

Обработка данных, построение трансляторов, вопросы программирования

Труды экономического факультета LIX

На английском языке

Vastutav toimetaja I. Amitan
Keeleline toimetaja M.-A. Laane

Kinnitanud TPI Toimetiste kolleegium 27.11.85

Trükkida antud 12.08.1986
MB-06154
Formaat 60x90/16
Trükipg. 7,75+0,25 (lisa). Arvestuspg. 6,1
Trükiarv 400
Tellimuse nr. 377
Hind 90 kop.
Tallinna Polütehniline Instituut, 200108 Tallinn, Ehitajate tee 5
TPI rotaprint, 200006 Tallinn, Koskla 2/9

Hind 90 kop.