tanulmányok

182/86

MTA Számítástechnikai és Automatizálási Kutató Intézet Budapest

Computer and Automation Institute Hungarian Academy of Sciences
Magyar Tudományos Akadémia Számítástechnikai és Automatizálási Kutató Intézete

PROCEEDINGS OF THE JOINT BULGARIAN-HUNGARIAN WORKSHOP ON "MATHEMATICAL CYBERNETICS AND DATA PROCESSING"

Scientific Station of Sofia University, Giulečica (Bulgaria), May 6-10, 1985

V O L II

Studies 183/1986
Tanulmányok 183/1986

A kiadásért felelős:

Dr. KEVICZKY LÁSZLÓ

Editors:
Szerkesztők:

J. DENEV,
B. UHRIN

Főosztályvezető:

DEMETROVICS JÁNOS

ISBN 963 311 212 5
ISSN 0324-2951

F O R E W O R D

The second Bulgarian-Hungarian Joint Workshop entitled *"Mathematical Cybernetics and Data Processing"* was held at the Scientific Station of Sofia University *"Giulečica"* between May 6-10, 1985. About 30 researchers from the following institutions attended the workshop:

- Centre of Mathematics and Mechanics of the Bulgarian Academy of Sciences, Division of Foundations of Cybernetics & Control Theory, Laboratory of Mathematical Linguistics;
- Computer and Automation Institute of the Hungarian Academy of Sciences, Division of Computer Science;
- Blagoevgrad Pedagogical Institute, Blagoevgrad;
- Central Institute of Computer Technics, Sofia.

Although the workshop lasted a few days only and the number of participants was not too large, an intensive work went on.

The themes involved were from the field indicated by the title of the workshop.

The papers presented at the workshop can be classified into four main subfields as follows:

- mathematical cybernetics;
- mathematical linguistics;
- computer and software architecture;
- data bases.

The present proceedings contains papers presented at the workshop. Because of the great number of papers, we divided the proceedings into two volumes, the first containing papers from the field of math. cybernetics and math. linguistics and the second one from those of comp. and software architecture and data bases.

The beautiful surroundings of the station *"Giulečica"* (the Rila mountains) and an excellent weather contributed substantially to the success of the workshop. The same can be said about the station itself.

We thank everybody who took part in the organisation of this pleasent and succesful meeting.

The organizing committee

ПРЕДИСЛОВИЕ

Вторая рабочая конференция "МАТЕМАТИЧЕСКАЯ КИБЕРНЕТИКА И ОБРАБОТКА ДАННЫХ" состоялась в Научной станции Софийского университета "Ιюлечица" с 6 по 10 мая 1985 г. В работе конференции участвовало свыше 30 научных работников из

- Единого центра математики и механики Болгарской Академии наук: сектор "Основы кибернетики и теории управления" и лаборатория "Математическая лингвистика" - организаторы конференции;
- Исследовательский институт вычислительной техники и автоматизации Венгерской Академии наук;
- Высший педагогический институт, Благоевград;
- Центральный институт вычислительной техники, София;

Хотя время конференции было ограничено, а число участников не очень большое, была проведена весьма интенсивная работа, результатом которой является настоящий сборник докладов конференции.

Тематика работы следовала направлению сотрудничества между Институтом математики БАН и ИИВТА ВАН по теме "Математическая кибернетика и обработка данных", в рамках которой проводилась эта встреча. Работы можно объединить в следующие группы:

- математическая кибернетика;
- математическая лингвистика;
- компьютерные и софтверные архитектуры;
- базы данных.

Этот сборник содержит в себе статьи прочитанные на конференции. Число статей было так велико, что сборник надо было разделить на два тома. Первый том содержит статьи принадлежащие к мат. кибернетике и мат. лингвистике, а второй - принадлежащие к компьютерной и софтверной архитектуре и базам данных.

Великолепная картина Рилских и Пиринских гор и прекрасная погода принесла весьма ощутимый вклад в успех конференции, которая заключилась не только в богатой программе, но и в установлении и поддержке дружественных контактов, которые происходили везде на заседаниях, во время прогулок и товарищеских встречах.

Внимательная забота персонала Научной станции обеспечила все условия и можно сказать даже комфорт для проведения работы встречи.

Благодаря коллегам из Благоевграда была получена возможность устроить экскурсию в один из самых красивых уголков Болгарии - горы Пирин.

Оргкомитет

C O N T E N T С О Д Е Р Ж А Н И Е

V O L II

COMPUTER AND SOFTWARE ARCHITECTURE

DATA BASES

COMPUTER AND SOFTWARE ARCHITECTURE

КОМПЬЮТЕРНЫЕ И СОФТВЕРНЫЕ АРХИТЕКТУРЫ

MTA SZTAKI Tanulmányok 183/1986 pp 11-17

AN ALGORITHM FOR SOLUTION OF RESOURCE PROBLEM IN THE AUTOMATIC SYNTHESIS OF HORIZONTAL MICROPROGRAMS

A. PETKOV, L. MANASIEV

Institute of Mathematics with Computing Centre
Bulgarian Academy of Sciences

One of the most perspective approaches to horizontal microprograms creation is to describe the control algorithms as a sequence of statements of a general purpose, architecturally independent algorithmic language. The described algorithms are translated into an equivalent sequence of microoperations - sequential microcode. However, no possibilities for concurent execution of some of the microoperations are reflected in it. This does not allow effective usage of the horizontal microinstruction potential for concurrent execution of several microoperations. Thus, it naturally leads to the problem of sequential microcode conversion into an equivalent concurent microcode. It is known under the name of "microcode compaction". The used compaction method fixes the resource, time and format dependences between the different microoperations. All mentioned investigations are characterized by the assumption that the resources preserve intact their state until it is changed by some microoperation. However, there are resources in the microarchitectures which could preserve their contents only for a definite period of time. These resources are known as "transitory data resources". The transitory data resources problem is well stated in (I), but no solution is provided.

A method for sequential microcode compaction in architectures containing transitory data resources is presented in the paper.

Let's assume that M= $MOP_1,\ldots,MOP_i,\ldots,MOP_j,\ldots,MOP_k$ is the sequential source microcode. Let's assume also that MOP_i microoperation updates the contents of the transitory data resource T, and the correct MOP_j execution requires the contents T defined by MOP_i. Thus, MOP_i and MOP_j are strongly limitted in terms of their participation in the compaction process. Hence, their allocation in the microinstructions in the resultant concurrent microcode should provide the defined contents T during MOP_j activation.

D I: "dtd" (data transitory dependence) relation is set between MOP_i and MOP_j, if the following conditions are satisfied for at least one transitory data resource T:

1. The resource T is updated by MOP_i.

2. The available contents t before the updating of MOP_j is is required for the correct MOP_j execution.

3. There is no MOP_k element from M, i.e. $i<k<j$ and MOP_k updates the contents T.

Let's assume that there are 'n' transitory data resources in the microarchitecture the M microcode is destined for. Without any limitation, they could be numbered and a sequentially numbered set must be generated $T_1,\ldots,T_n$. On this basis, two vectors can be connected with each M element: TWV (transitory write vector) and and TRV (transitory read vector):

- TWV = $(x_1,\ldots,x_n)$, where

$$x_i = \begin{cases} 1 \text{ (true), if } T_i \text{ satisfies p.I from D I.} \\ 0 \text{ (false), otherwise.} \end{cases}$$

- TRV = $(y_1,\ldots,y_n)$, where

$$y_i = \begin{cases} 1 \text{ (true), if } T_i \text{ satisfies p.2 from D I.} \\ 0 \text{ (false), otherwise.} \end{cases}$$

The couple (TWV,TRV) which is compared with every microoperation from M represents its potential according to dtd relation. It could be used for strict determination of the place of a M element in the existing M chains of dtd-dependences. The following classification is used for its definition:

a. $M1 = \{ MOP_i / MOP_i \in M \text{ and } TWV_i = \bar{0},\ TRV_i \neq \bar{0} \}$

b. $M2 = \{ MOP_i / MOP_i \in M \text{ and } TWV_i \neq \bar{0},\ TRV_i = \bar{0} \}$

c. $M3 = \{ MOP_i / MOP_i \in M \text{ and } TWV_i = \bar{0},\ TRV_i = \bar{0} \}$

d. $M4 = \{ MOP_i / MOP_i \in M \text{ and } TWV_i \neq \bar{0},\ TRV_i \neq \bar{0} \}$

It is evident that $M = \bigcup_{i=1} M_i$ and $M_i \cap M_j = 0$ for $i,j = 1 \div 4, i \neq j$. The elements of the different classification classes have the following logical meaning:

- $MOP_i \in M$ .MOP_i can only end a chain of dtd-dependences in M, but it cannot be the first or middle member of the chain. This is due to the fact that it is not possible to exist $MOP_j \in M$, $i \neq j$ and MOP_i dtd MOP_j.

- $MOP_i \in M2$.MOP_i can lead to the formation of a chain of dtd-dependences but it is possible for it to be only the first member of the chain.

- $MOP_i \in M3$.MOP_i cannot be a member of the chain of dtd-dependences.

- $MOP_i \in M4$, since this is the general case, nothing can be said about these elements.

It is possible to perform analysis in the M set in order to establish the dtd relations, existing between its elements. This process is called dtd analysis. Its objective is to build the following oriented loaded graph:

- vertexes : M elements
- edges: there is an edge MOP_i.MOP_j if between the corres-

ponding vertexes MOP_i dtd MOP_j.

- load: a chain of natural numbers is assigned to the edge $MOP_i.MOP_j$ containing the indicese of those transitory data resources which cause the appearance of dtd relation between MOP_i and MOP_j.

The following assertions about the relation between the stated classificatioh classes and the dtd graph have been proved:

AI: The roots of the essential subgraphs are elements of the M2 class.

A2: The single vertexes in the dtd graph are elements of M2UM3 and all M3 elements are single vertexes.

A3: The MI elements are leaves in the essential subgraphs of the dtd graph.

The following characteristics of MI,M2,M3 and M4 classes could be defined on the basis of AI, A2 and A3:

- MI - leaves in the essential subgraphs of the dtd graph
- M2 - roots of the essential subgraphs or single vertexes
- M3 - single vertexes in the dtd graph
- M4 - intermediate elements or leaves in the essential subgraphs of the dtd graph.

The following assertion, allowing decreasing the algorithm steps is very important for the development of an efficient dtd analysis algorithm.

A4: If MOP_i dtd MOP_j according to the transitory resources $T^i = \{ T_{i1}, \ldots, T_{is} \}$, then for any other MOP_p , i.e. $p \neq i, p < i$ and MOP_p dtd MOP_j according to resources $T^p = \{ T_{p1}, \ldots, T_{ps} \}$, follows $T^i \cap T^p = 0$.

On the basis of the above assertions the following iterative dtd analysis algorithm is defined:

Step A. A characterizing vector couple (TWV,TRV) is created for every M element.

Step B. MI,M2,M3,M4 classes are defined.

Step C. MI∪M4 elements are ordered in a descending order of their indices.

Step D. The assignment P:=M2 is performed for the working set P.

Step E. The P elements are ordered in a ascending order of their indices.

Step F. The P elements are sequentialy processed as follows: Let MOP_a be a consecutive element. MOP_a is checked according to TI criterion for the existence of dtd relation with all MOP_j, i.e. $MOP_j \in MI \cup M4$ and $a < j$. When such a relation is determined, in the corresponding TRV_j vector all components which comform to the transitory data resources causing the dtd relation are set to zero.

Step G. Has P been exhausted? If "NO",go to step F.

Step H. The set $Q = \{ MOP_c / MOP_c \in M4$ and $MOP_a \in p$ is formed $\}$ If $Q \neq 0$, then P:=Q and go to step E.

In the initial step execution, M2 elements, which are root s of essential subgraphs are separated from the single elements. In the next passes, the probable intermediate elements of the essential subgraphs are manipulated with. T2 provides strict information about the loading of the defined dtd connections by means of the non-zero vector components $\bar{r} = \bar{z}$&

$$\neg((\bigvee_{k=j+1}^{j} TWV_k) \;\&\; \bar{z}) \quad \text{/specially for } MOP_i \text{ dtd } MOP_j\text{/}$$

The received in the given paper theoretical results and the developed on their basis dtd analysis algorithm allow to separate the microoperations groups, appearing due to the existence of transitory data resources in the microarchitectures. The elements defined as vertexes of one and the same essential subgraph must be correctly synchronized which could be performed without any principle dificulties. Further on in the compaction process, the so synchronized elements are defined as an enlarged microoperation So, the source M set is defined as a M' set of enlarged single elements. The well-known principles of local and global compaction are valid and naturaly applied for M' while the existence of the transitory data resources is ignored.

REFERENCES

I. Landskov D., Davidson S., Some experiments in local microcode compaction for horizontal machines, IEEE Trans. on Comp. Vol. C-30, I98I, 7.

АЛГОРИТМ РЕШЕНИЯ РЕСУРСНОГО ПРОБЛЕМА ПРИ АВТОМАТИЧЕСКОМ СИНТЕЗЕ ГОРИЗОНТАЛЬНЫХ МИКРОПРОГРАММ

А. Петков, Л. Манасиев
Институт математики с ВЦ - БАН

В статье рассматривается метод уплотнения последовательного микрокода для микроархитектуры, содержащей ресурсы с "временными данными". Даны основы теоритического аппарата, позволяющего нахождения этих групп микроопераций, требующими синхронизации по отношению их ресурсов с временными по способу требуемому соответствующими ресурсами с временными данными.

Рассмотрен также алгоритм, определяющий различные группы микроопераций, зависящих от одних и тех же ресурсов с временными данными.

MTA SZTAKI Tanulmányok 183/1986 pp 19-25

ALGORITHM FOR PRELIMINARY GROUPING AND TRANSFORMATION OF INSTRUCTIONS IN DATAFLOW GRAPH AND TRANSFORMATION RULES

N.K. DABOV

Central Institute for Computing Technics, Sofia

INTRODUCTION

A programmer can easily express his object of computation and prove correctness of the program through functional programming style, proposed by Backus [1]. The von Neumann computer architecture is not appropriate for this new style of programming whereas the dataflow concept [2] is a candidate for effective executor of functional program exploiting parallelism embedded in the program. Unfortunately, dataflow machines are not so effective when executing programs with low level of parallelism. The paper presents an algorithm for preliminary grouping and transformation of instructions in the dataflow graph and some transformation rules. It is hoped that this algorithm will reduce the communication overhead, and will increase the effectiveness of the dataflow machine for programs with low level of parallelism. Besides, an opportunity is created for control of machine resources.

ALGORITHM FOR PRELIMINARY GROUPING AND TRANSFORMATION

In dataflow machines instructions can be executed as soon as all their input operands have arrived. A dataflow program is viewed as a graph made of nodes interconnected by arcs. The arcs can carry tokens bearing value. A node is enabled when all its input arcs carry tokens. Nodes are simple operations (addition, substraction, etc.) manipulating with one or two simple operands. Due to the fact that the object of computation is divided into large number of small tasks, communication overhead is introduced

in order to link tasks with both their predecessors and successors. In the case when the resources for processing of instructions and data communications between processors are limited, this overhead may degrade the expected performance. Another disadvantage of dataflow machines is their low performance when executing programs with low level of parallelism. Some authors suppose that the von Neumann computer architecture is more efficient for execution of such programs [3,4]. For these reasons,an appropriate grouping of instructions will lower the communication overhead and a better performance under low parallelism can be obtained while permitting parallelism to be exploited at a higher level [4].

On the other hand, part of the dataflow graph can be transformed into more efficient form without change of its meaning. In other words, the dataflow graph can be reshaped before data arrival using some graph transformation rules. The preliminary transformation reduces the number of instructions in the dataflow graph if it is possible. The graph transformation rules will be discussed in the next section.

The grouping of instructions and the preliminary transformation of the dataflow graph can be combined into one process, which is performed dynamically, at run-time. This process is carried out by special processors for preliminary grouping and transformation (PGT) which work concurrently with the data processing elements (DPE). Each PGT performs preliminary grouping and transformation referring to the rules stored in its local storage. In fact, all PGT's execute one and the same program for optimization of parts of the dataflow graph which are assigned to them. The algorithm of this optimization program is as follows:

-Step 0. PGT waits until part of dataflow graph is assigned to it. If so, executes step 1.
-Step 1. PGT reads the assigned to it part of the graph. Analyzes the instructions contained within this part of the graph. If preliminary transformation is needed executes step 2, else step 3.
-Step 2. Preliminary transformation is performed referring to the transformation rules (see below).
-Step 3. The part of the dataflow graph is transformed into sequential, von Neumann code.
-Step 4. A new identifier is assigned and as a result an actor is created. Then PGT performs the corresponding address changes in those actors and instructions that generate input operands for this actor.
-Step 5. The actor is stored.
-Step 6. PGT informs for the end of preliminary grouping and transformation. Executes step 0.

This algorithm for preliminary grouping and transformation is independent from the programs executed by the dataflow machine. Since graph transformation process (step 3) is expected to require fairly long time, the transformation rules may be different for the different classes of programs or language systems.

TRANSFORMATION RULES

As was mentioned above, the preliminary transformation reduces the number of instructions in the dataflow graph if possible. The transformation rules can be based on algebriac and Boolean laws, and on preliminary computations using already arrived input operands. Some rules may depend on the chosen language system.

1) TRANSFORMATION RULES BASED ON ALGEBRIAC AND BOOLEAN LAWS

These rules are based on commutative, associative, and distributive laws and refer to operations that are included in all language systems. Since this transformation does not depend on the arrival of data tokens, we will call it "static".

1-1) Changing the order of operators

Examples: A*B + A*C = A*(B+C)

NOT U AND NOT V = NOT(V OR U)

1-2) Inverse functions

Two inverse functions can be deleted from the graph if they are to be executed one after the other. The exponential and the logarithmic functions, sin and arcsin are examples.

1-3) Algebriac laws

Examples: $(e^{A})^{B} = e^{A*B}$

$\ln A / \ln B = \log_{B} A$

$2*\sin A*\cos A = \sin(2*A)$

2) TRANSFORMATION RULES BASED ON ALREADY ARRIVED INPUT OPERANDS

When PGT reads part of dataflow graph, some of the input operands for this part of the graph may have had arrived as shown in Fig. 1a. A black dot on the arc reflects the presence of input operand.This part of the graph can not be reduced using the rules of static transformation. Nevertheless, it can be reduced when taking into account the presence of the arrived input operands. In this example a preliminary computation (A+C) can be made and the reduced graph is shown in Fig. 1b. Another example is the preliminary execution of SWITCH operator when only the predicate input operand is present. The transformation based on already arrived input operands we call "dynamic".

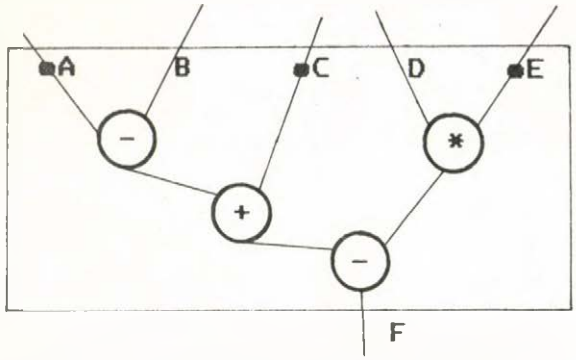


Fig 1a Part of graph before transformation

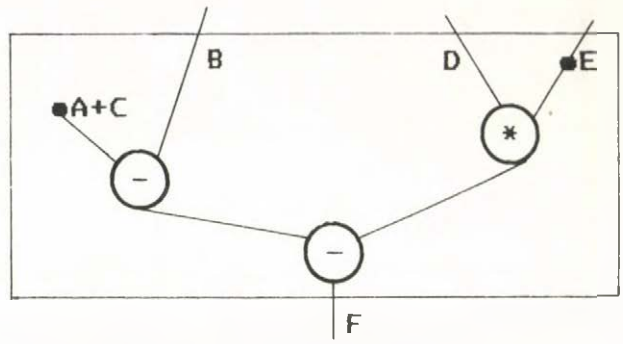


Fig 1b Same part after transformation

3) TRANSFORMATION RULES BASED ON LANGUAGE SYSTEM

The sequence of functions can be reordered using transformation rules which depend on the law of composition defined in a language system. These rules are different for the different languages. An example of such "specialized" transformation rules based on APL primitive operators is shown in [5].

CONCLUSION

The preliminary transformation reduces the number of instructions in a dataflow graph and contributes to improvement of machine performance for different classes of programs or languages. As a result from the grouping of instructions, the machine becomes a hybrid dataflow/von Neumann system combining the advantages of the two systems. Since the grouping of instructions is performed dynamically, at run-time, an opportunity for effective control of the machine resources is created.

REFERENCE

[1] Backus, J., Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs, Comm. ACM 21,8,Aug.1978

[2] Treleaven, P.C., et.al., Data-Driven and Demand-Driven Computer Architecture, Computing Surveys, 14-1, Mar.1982

[3] Gajski, D.D., et.al., A Second Opinion on Data Flow Machines and Languages, Computer, 15-2, Feb.1982

[4] Guadiot, J.L., Ercegovac, M.D., Performance Analysis of a Data Flow Computer with Variable Resolution Actors, Proc.4th Int. Conference on Distributed Computing Systems, May 1984

[5] Kubo, M., Kohmoto, T., Ohno, Y., A Data Flow Machine with Optimization Driven Graph Reduction Mechanism, Proc.4th Int. Conference on Distributed Computing Systems, May 1984

Алгоритм предварительного группирования и трансформирования инструкции потокового графа и правила трансформирования

Дъбов Н.К.

Центральный институт вычислительной техники - София

Р Е З Ю М Е

В статье предлагается алгоритм предварительной обработки и группирования инструкций в потоковом графе, которые производятся динамично в процессе работы. Этот алгоритм дает возможность уменьшить коммуникации и увеличить эффективность потоковой машины при выполнении программы с низкой степенью параллелизма. Описаны правила предварительного трансформирования графа.

MTA SZTAKI Tanulmányok 183/1986 pp 27-33

ПРОГРАММНАЯ СИСТЕМА ОЦЕНКИ ПРОИЗВОДИТЕЛЬНОСТИ МАТРИЧНОГО ПРОЦЕССОРА ЕС2706

А.ПЕТКОВ, А.ТЕРЗИЕВ, В.АНГЕЛОВ, Д.ВЪНДЕВ
Институт математики с ВЦ
Болгарская академия наук

Предметом настоящего доклада является программная система оценки производительности МП ЕС2706 по отношению к номинальной производительности ЭВМ ЕС, в составе которой она функционирует. Оценка производительности совершается следующим способом:

- задан комплекс представительных программ, реализирующих типичные приложения МП ЕС2706;
- на базе алгоритмов, заложенных в эти представительные программы, реализован аналогичный комплекс программ, функционирующих однако на центральным процессором;
- созданы специальные управляющие программы, которые подготовляют, активируют и следят за исполнением представительных программ;
- с помощи управляющих программ осуществляется подготовка и генерация подходящих входных данных для всех представительных программ, причем таким образом формируются нагрузки для МП и ЦП;
- хронометрируется выполнение нагрузок отдельно для каждого из процессоров:

Хронометрирование нагрузки в центральном процессоре совершается при помощи учета показания таймера непосредственно перед активированием подготовленной нагрузки и сразу после окончания ее работы. Так как представительные программы для центрального про-

цессора не содержат входно-выходных операций, то в случае когда программы находятся в реальной памяти операционной системы и в то же время нет другого активного задания с более высоким приоритетом, чистое время центрального процессора будет совпадать с астрономическом временем при выполнении нагрузки.

Хронометрирование нагрузки для матричного процессора более сложна, т.к. в таком случае и процесс этот разнороден. Активизирование представительной программы совершается в центральном процессоре, но после подготовительных операций, совершенных в центральном процессоре, основная обработка нагрузки исполняется в матричном процессоре и наконец работа по нагрузке заканчивается опять в центральном процессоре. Связь между центральным и матричным процессорами осуществляется при помощи механизма входно-выходного обмена и чистое время центрального процессора составляет только небольшую часть полного времени, необходимого для обработки нагрузки в системе. Для нагрузки матричного процессора хронометрирование совершается при помощи четырехкратного учета показания таймера: Непосредственно перед активированием нагрузки; после окончания подготовительных работ в центральном процессоре ,непосредственно перед активированием основной работы, совершенной матричным процессором;сразу после окончания основной обработки в матричном процессоре; сразу после окончания работы над всей нагрузкой. Таким образом формируются два временные интервала для для работы всей нагрузки: первый включает подготвительные работы в центральном процессоре и основную обработку в матричном процессоре, а второй – только чистое время основной обработки над нагрузкой, совершаемой в матричном процессоре.

- при помощи управляющей программы дефинируется и обрабатывается довольно большая статистическая совокупность нагрузок;

- сопоставляют результаты хронометрирования одних и тех же нагрузок обоих процессоров и вычисляется сравнительная производительность по отношению к номинальной производительности центрального процессора.

Программная система включает два аналогичных комплекта представительных программ - один для центрального процессора и другой для матричного процессора. Каждый комплект состоит из 179-ти программ. Дальше будем называть эти программы нагрузочными функциями. Они реализуют следующие типичные применения МП ЕС2706:

- базовые векторные арифметические операции (46 функции);
- операции типа вектора-скалара (17 функций);
- сравнение векторов (13 функций);
- операции над комплексными векторами (25 функции);
- матричные операции (12 нагрузочных функций);
- быстрое преобразование Фурие (6 нагрузочных функций);
- специальные операции (6 нагрузочных функций);
- современные математические методы (30 функций);
- обработка сигналов (24 функций).

Как основные входные данные образования нагрузок используются идентификаторы нагрузочных функций, но кроме них для образования нагрузок необходима и дополнительная информация.

Чтобы обеспечить образование и выполнение требуемой совокупности нагрузок создан специальный управляющий язык системи оценки производительности МП ЕС2706. При помощи операторов управляющего языка определяется входный поток описаний нагрузок, которые обрабатываются и выполняются под управлением системы. Входной поток операторов управляющего языка системы указывает также на режим работы системы, на распределение ресурсов в системе и на

информацию о номинальной производительности центрального процессора над которым будет работать система.

Входные данные о работе нагрузочных функций можно задать извне при помощи описания нагрузки в соответствующем операторе или генерируются автоматично внутренне системой.

Определение и активирование нагрузочных функции осуществляется при помощи операторов:

```
Control = MPTEST [,Count = {1/n}] [,Dump = {No/Input/Output/ALL}
[,Performance = {0/n}] [,Execution = {Apu/Cpu/Both}]
[,Initialgen = {0/n}]
name1 [([Count = {1/n}] [,Dump = {No/Input/Output/ALL}]
  [,Initialgen = n] [,Execution = {Apu/Cpu/Both}]
  [, Pi = {Vi / (ni, mi, vi1, vi2 ... vi nxmi)}])]
name2 [(.....)]
. . . . . .
namek [(.....)]
END
```

Параметры определяют соответно: число выполнений нагрузочной функций (COUNT); тип информации, которая накапливается в файле данных (DUMP); направление товара (EXECUTION); начальная стойность генератора случайных чисел, используемый при автоматическом генерировании данных (INITIALGEN); производительность центрального процессора (PERFORMANCE); имя нагрузочной функций (name 1...K); определение извне стойности i-ого по порядку параметра нагрузочной функций (P_i).

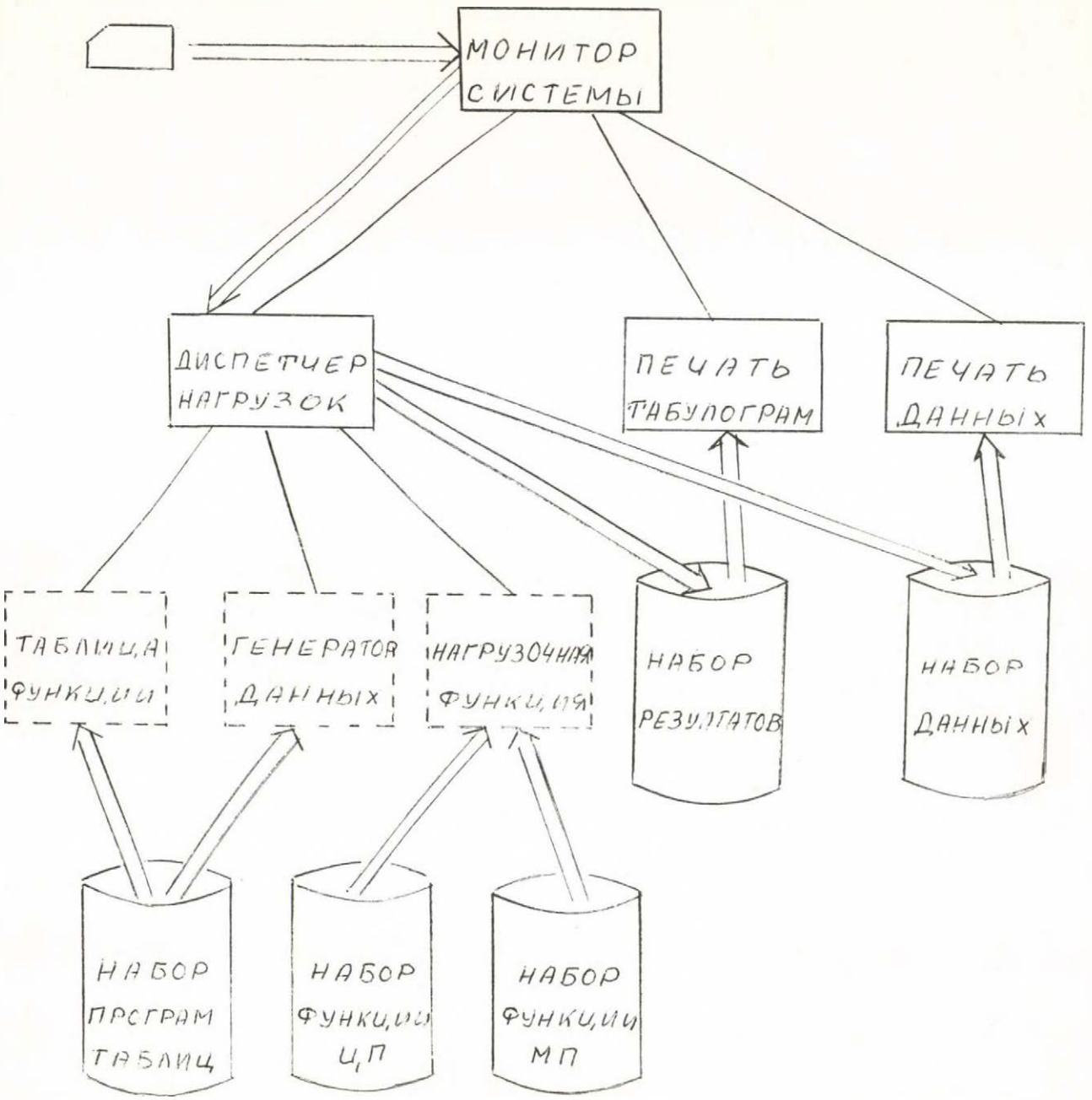
При распечатки данных можно сравнить результаты выполнения каждой функции поотдельно над двумя процессорами путем поэлементного сравнивания соответствующих массивов с заданной точностью. Если обнаружено хотя бы одно несоответствие в элементах ,результаты выполнения функции считаются неадекватными и выводится соответствующее сообщение об этом.

Основным результатом работы системы является табулограма, в которой в табличном виде печатаются временные интервалы, измеренные при выполнении отдельных нагрузок, коефициенты ускорения для МП ЕС2706 и относительная производительность МП ЕС2706.

Общая структура системы и потоки информационного обмена между отдельными компонентами показаны на фиг.1. Система работает под управлением операционных систем для ЕС ЕВМ с виртуальной памятью: ОС ЕС или ОС/351 с генерированным методом доступа APEX для МП ЕС2706. Программы разработанны на разных языках в зависимости от их специфики:

- монитор и диспетчер нагрузок на языке Асемблера для ЕС;
- программы печати на ПЛ/1;
- набор 179 программ, реализующих разных нагрузочных функций, на языке Фортране.

Система показала высокую надежность при эксплуатации.


МОНИТОР СИСТЕМЫ
ДИСПЕТЧЕР НАГРУЗОК
ПЕЧАТЬ ТАБУЛОГРАМ
ПЕЧАТЬ ДАННЫХ
ТАБЛИЦА ФУНКЦИИ
ГЕНЕРАТОР ДАННЫХ
НАГРУЗОЧНАЯ ФУНКЦИЯ
НАБОР РЕЗУЛЬТАТОВ
НАБОР ДАННЫХ
НАБОР ПРОГРАМ ТАБЛИЦ
НАБОР ФУНКЦИИ Ц.П
НАБОР ФУНКЦИИ МП

Фиг. 1

A PROGRAM SYSTEM FOR MEASURING THE PERFORMANCE OF THE EC2706 ARRAY PROCESSOR

A.Petkov, A.Terziev, V.Angelov, D. Vandev
Institute of Mathematics with Computer Center, BAS

Abstract

The architecture and principles of a program system for measuring of the EC2706 array processor are considered. An approach for the definition and generation of data for the measurement is described. A method and a mechanism of passing parameters from one kind of program module to others and their transfer between the CPU and array processor are presented.

MTA SZTAKI Tanulmányok 183/1986 pp 35-41

AN ANALYTIC MODEL FOR THE CONGESTION CONTROL STUDY IN A GATEWAY NODE

S. DIMITROV

Institute for Computing Techniques
Bulgaria

1. Introduction

The interconnection of local area networks (LANs) with other LANs or wide area networks enlarges the access to a wider range of local and remote resources [1,3]. The incompatibilities in the transmission media and in the protocol layers are to be solved when a LAN is connected to a diverse network. This is done in a special node which acts as an interface between the two networks and is called a gateway [5].

The gateway is a "shared resource" among all the users in the two networks. For this reason heavier traffic is expected to pass through it than through the normal stations so it may become the "bottleneck" in the internet [1]. That is why congestion control is necessary which means maintaining the input traffic to a subsystem (e.g. a gateway) within limits compatible with the amount of resources available to the subsystem [4].

The congestion probabilities in a gateway between a LAN of Ethernet-type and a low speed buffer insertion ring LAN is studied in this paper. The gateway is presented as a queueing system with losses of $M/M/1/N<\infty$ type. The maximum rate of the internet traffic and the buffer space necessary for congestion avoidance are evaluated.

2. The gateway as a queueing system with losses

The gateway node studied is intended to connect a 10Mb/s

carrier sense multiple access with collision detection (CSMA/CD LAN to a 125kb/s- buffer insertion ring LAN with station priority [2]. From the great difference in the tansmission speeds of the two LANs it becomes obvious that one of the major problems in the implementation of such a gateway would be the control of the traffic rate in direction Ether-type LAN to the ring for congestion avoidance.

An $M/M/1/N<\infty$ queueing system with losses and a limited queue is an excellent tool for the preliminary evaluation of the congestion probabilities in the gateway. The input process is Poisson with arrival rate of the calls $\lambda > 0$. This is the internet traffic from the Ether-type LAN to the ring. The service centre (SC) presents the CPU for internet packets' processing and the transmission channel fo the ring. The service time is exponentially distributed with mean rate $\mu>0$ and is defined by the mean time for packet processing and the mean time for its transmission to the ring. Station priority means that a station (i.e the gateway) having a transmit request pending is allowed to transmit its frame(s) immediately if there is no transit frame being sent from the insertion buffer. Otther wise, it must defer its transmission until the end of the transit frame, currently being transmitted. That is why two values of μ are considered: $\mu_g = 250$ for a free channel and $\mu_g = 100$ for a busy channel. There are N places for waiting: N-1 in the queue and 1 in the SC. One place in the queue means the mean buffer space for an internet packet.

The stationary probability k calls to be found in the system

$$P_k = (1-\rho)\rho^k/(1-\rho^{N+1}) \qquad ,k=0,1,\ldots N \quad (1)$$

$$\rho = \lambda/\mu$$

From Eq.1 the probability a call to be rejected, i.e congestion to occur in the gateway is derived:

$$P_N = (1-\rho)\rho^N/(1-\rho^{N+1}), \rho \neq 1 \quad (2a)$$

$$P_N = 1/(N+1), \quad \rho = 1 \quad (2b)$$

The probability the SC to be found busy is:

$$\tilde{\rho} = \rho(1-\rho^N)/(1-\rho^{N+1}), \rho \neq 1 \quad (3a)$$

$$\tilde{\rho} = N/(N+1), \quad \rho = 1 \quad (3b)$$

The results from the solving of eqs.2 and 3 for different values of λ and N-1 are summarized in tables 1-4.

Table1

Results for p_N,% (μ_g=250)

N-1 / λ	0	1	2	3	5	10	15	100
25	9	0,09						
50	16	0,5						
100	14	10,2			0,49			
250	50	33			14	8,3		
500	66	57	53	51	50	50	50	
1000	80	76			75	75	75	
2000	88	87			87	87	87	

Table 2

Results for p_N,% (μ_3=100)

N−1 / λ	0	1	2	3	5	10	15	100
25	20	1,2						
50	33	6,2						
100	50	33			14			
250	71	65	61,5	60,6	60	59	59	59
500	83	81			80	80	80	80
1000	91	90			90	90	90	90
2000	95	95			95	95	95	95

Table 3

Results for $\tilde{p}$,% (μ_3 =250)

N−1 / λ	0	1	2	3	4	5	10	15
25	9	10						
50	16	20						
100	30	35				41		
250	50	66				86	92	
500	66	85	93	97	98	98	99	99
1000	75	95	98			99	99	99
2000	88	98	99			99	99	99

Table 4

Results for $\tilde{p}$.% (μ_g =100)

λ \ N-1	0	1	2	3	4	5	10	15
25	20	25						
50	33	46						
100	50	66						
250	71	89	96	98	99	99	99	99
500	83	96				99	99	99
1000	91	99				99	99	99
2000	95	99				99	99	99

The results in tables 1-4 show that for arrival rates λ=500 (μ_g=250) and λ=250 (μ_g=100) and a queue size of 3-4 places the two probabilities saturate at values 50-60% for a call rejection and 99% for a busy SC respectively. Further increase in the queues size makes no difference. It is evident that these values off the arrival rate are critical for the congestion control. So traffic rate should be kept below this level for congestion avoidance. E.g, almost congestion-free operation off the gateway may be guaranteed at λ=100 and 5 waiting places.

3. Conclusions

From the above assumptions the following conclusions can be drawn:

a. Endless increase in buffer space cannot solve the congestion problem.

b. The traffic rate from the higher speed LAN to THe lower speed LAN is to be controlled. E.g, the number of internet packet sent to the ring to be kept below certain threshold which will

depend on the buffer size and the service rate, i.e the traffic rate will be as high as the gateway can take up with. This is similar to the "isarithmic control"[4].

c.The throughput may be increased if the gateway is implemented with two CPUs each of which is assigned to one of the LANs or one off the directions.

4. Summary

The analytic model developed yielded the probabilities for arrived call to be rejected (i.e congestion to occur) and the probabilities for the SC to be found busy as a function of traffic rate and queue (buffer) size. Conclusions are made about the maximum traffic rate for congestion avoidance, the buffer size necessary and that control over the number of internet packets sent to the ring is needed.

5. References

1. Benhamou E.,J.Estrin,Multilevel Internetwork Gateways, Computer,Sept.83
2. Bux W.,M.Schlatter,An Approximate method for the performance analysis of buffer insertion rings,IEEE trans.on Communications,V.COM-31,No1,Jan.83
3. Danthine A.,Network Interconnection,Local Computer Networks (Ravasio,Hopkins,Naffah eds.),North Holland,1982
4. Pouzin L.,Flow Control in Packet Switched Data Networks, IEEE tr.on Communications,V.COM-29,No4,Apr.81
5. Sunshine C.,Interconnection of Computer Networks,Computer Networks,V.1,No3,Jan.77

МОДЕЛЬ С ОЧЕРЕДЯМИ ДЛЯ ИССЛЕДОВАНИЯ ПЕРЕГРУЗОК В МЕЖСЕТЕВОМ УЗЛЕ

Стефан Димитров
Центральный Институт Вычислительной Техники
бул."Ленин"VIIкм,1113София,Болгария

РЕЗЮМЕ

Межсетевой узел для соединения локальной вычислительной сети (ЛВС) типа Етернет с скоростью передачи 10Мв/s и кольцевой ЛВС с примыканием регистров с скоростью 125кв/s представлен как система типа M/M//1/$N<\infty$. Входящий поток вызовов представляет межсетевой трафик от ЛВС типа Етернет к кольцевой ЛВС. Центр обслуживания (ЦО) процессор обработки пакетов и каналь кольцевой ЛВС. Время обслуживания- среднее время обработки пакета и его передачи по кольцу. Места ожидания конечние- буферы хранения пакетов. Получены вероятности перегрузок узла и занятости ЦО в зависимости от места ожидания и интензивности трафика. Сделаны выводы для контроля перегрузок.

MTA SZTAKI Tanulmányok 183/1986 pp 43-49

TRANSPORT SERVICE IN LOCAL AREA COMPUTER NETWORK

A. PETKOV, A. TERZIEV, V. DIMITROV

Institute of Mathematics with Computing Centre
Bulgarian Academy of Sciences

The interaction between the remote processes on the distributed system are supported by the primitives of the computer network. The object of discussion of the present paper are the local-area computer networks designed for:

- building of integrated of personal computers, minicomputers, and specialized I/O devices;
- building of efficient interactive environment between the computer components.

We assume, that there will be few components with large external memory in the distributed system. While the other system components will use the resources of the more powerful components. Hence, on the basis of these assumptions, we have divided the computer components of the distributed system into two logical types:

- active components which use resources of other components;
- passive components which provide resources to other components with inadequate resources of their own.

The information exchange between the computers is provided by the local-area network. The structure of every component of the local-area network is given in Fig.I.

The combination of network stations, communication environment and the interface with the host systems we will call a basic local-area network. The structure and parameters of the

basic network are fixed ones and they are usually implemented by the manufacturer.

For the purposes of information flows maintenance in the distributed system, the basic network is tuned in the host systems by means of special software. Thus, a local-area computer network is built with an architecture corresponding to the given specification of the distributed system. The LAN architecture is given in Fig.2. Three levels could be distinguished: a basic network level, a common transport level and a high level with three parallel branches for three types of applications.

The aim of the given paper is the building of an interactive service on the common transport level of the local-area computer network (LACN) for a distributed system with the described features. As it has already been mentioned, we have divided the components into passive and active ones according to their interaction with the remaining components of the distributed system. The expansion of this division to the LACN transport level (Fig.3) leads to the creation of an asymetrical, unbalanced protocol where the virtual connections are in halfduplex mode.

The transport service primitives correspond to the ISO model but some simplifications have been carried out. The most important feature is that the primitives are divided into two groups: an active transport service and a passive transport service.

The active transport service comprises the following primitives:

- setting up of connection (CONNECT);
- terminating of the connection (DISCONNECT);
- sending of a message (SEND);
- expecting of a message (RECEIVE);

- activating of an inactive preset connection (ACTIVATE);
- disactivating of a preset connection (DISACTIVATE).

The passive transport service comprises the following primitives:

- expective passively the query for a connection or a message along a preset connection (LISTEN);
- confirmation of a query for a connection (ACCEPT);
- rejection of a query for connection (REJECT);
- sending of an expected message or confirmation of a received message (RESPONSE);
- confirmation of the cancellation of a given connection (CANCEL)
- waiting for sending of the required message (WAIT).

The interface between the high levels and the transport service is implemented by a special data structure, called a connection block. An independent block is built for each connection. The main block elements are:

- a local header preceding each message issued along the connection;
- remote header placed in the title of the last message received in that connection;
- connection parameters which contain the name of the host network station the connection belongs to; the name of the network station the host is connected to and which supports the connection block; the number of local process which uses the connection; the number of the remote process which uses the other side of the connection; the maximum allowable size of the messages transmitted along this connection; maintenance parameters including priority and secrecy code of the

connection;

- a list of transmission buffers containing the addresses and sizes of the buffers which contents is used for the formulation of the messages;
- a list of reception buffers containing the addresses and sizes of the buffers where every received message is distributed;
- return code containing the code of the result of the performed transport service activitities.

A characteristic feature of the connection block format is the capability of the transport service to transmit and receive every message, consisting of several independent parts.

The transport service implementation (Fig.4) is performed on three levels in order to provide its independence from the features of the basic network.

During the experimental implementation performed for an IZOT IO3IC personal computer, Assembler language was used in the programming of the first transport service level and Pascal was used for the second and third levels which provides the levels with definite machine independency. For basic LACN was be used the low-speed MULTILINK network.

R E F E R E N C E S

1. IS 7498 Information Processing Systems - Open Systems Interconnection - Basic Reference Model
2. IS 8348 Information Processing Systems - Data Communications - Network Service Definition
3. ISO 8802 Local Area Networks.

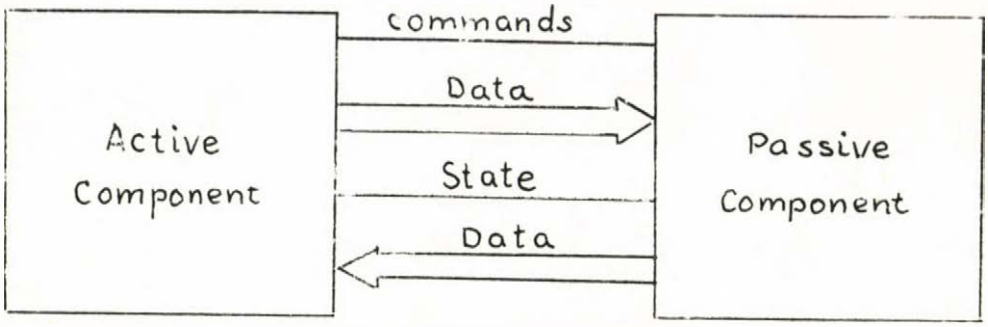


Fig.3. Local Area Network Partitioning

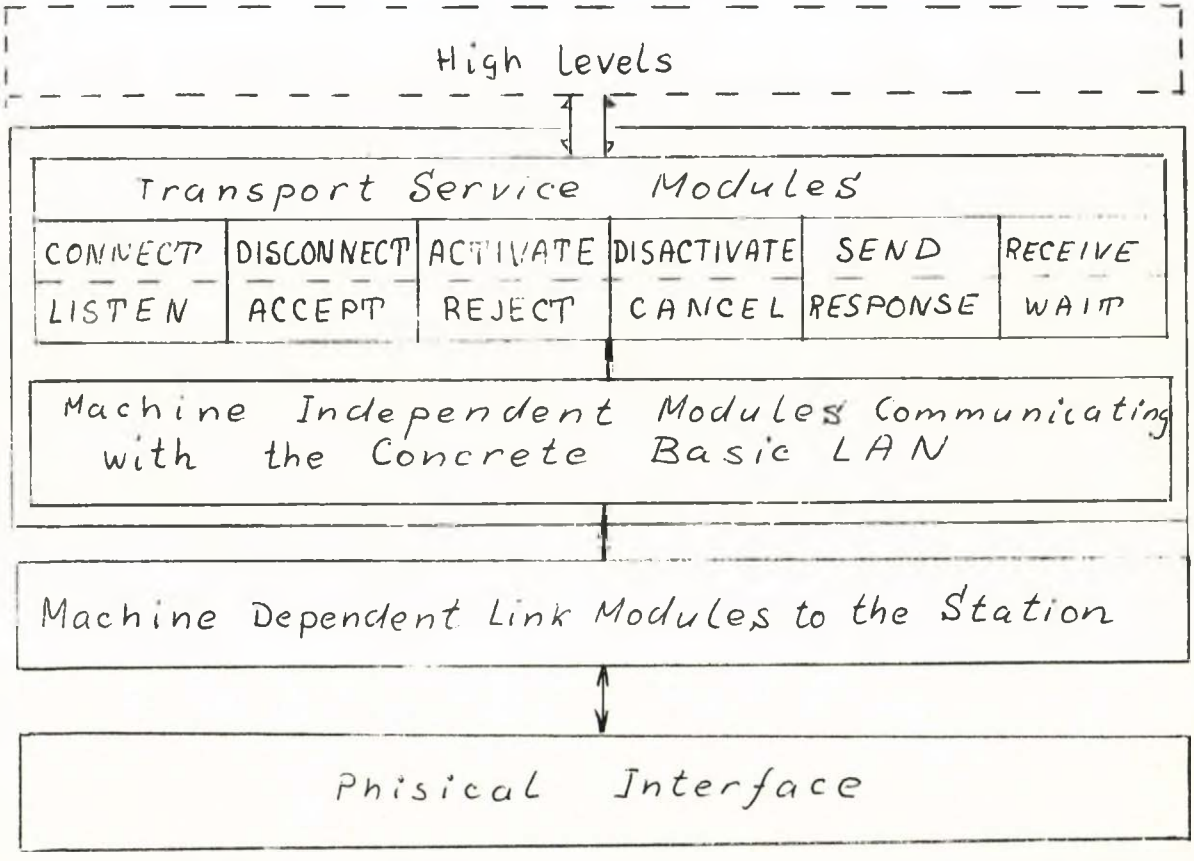


Fig.4. Programme's Modules Structure

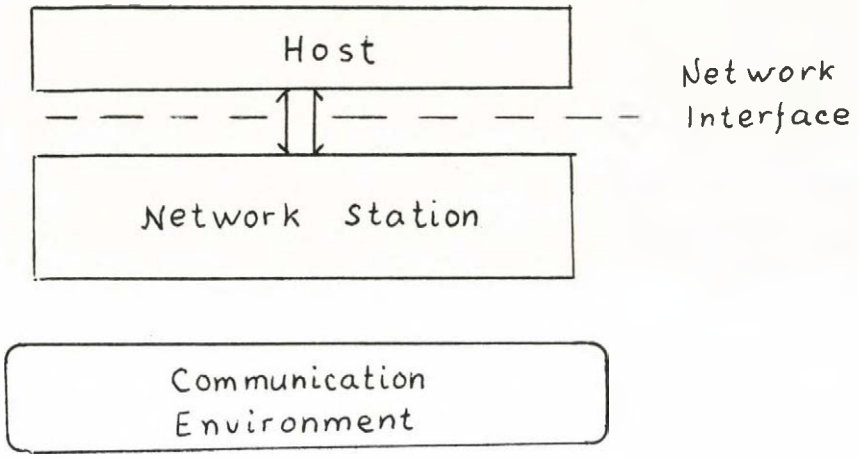


Fig. 1. Component Structure

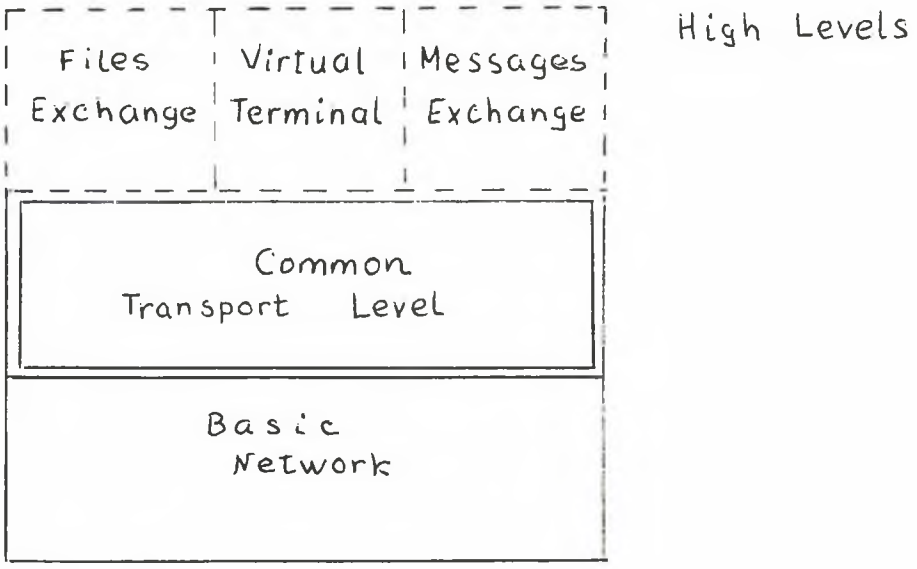


Fig. 2. Local Area Network Architecture

Транспортная служба в локальной вычислительной сети

А. Петков, А. Терзиев, В. Димитров
Институт математики с ВЦ - БАН

В статье предлагается транспортная служба, предназначенная для локальной сети в составе которой участвуют небольшое число компонентов с более мощными вычислительными ресурсами и значительно большое число компонентов с недостаточным количеством собственных ресурсов.

В такой сети совершено логическое разделение компонентов на два типа: активные, которые потребляют ресурсы и пасивные, которые предлагают ресурсы потребления.

На основе сделанного разделения, создана транспортная служба, которая различается для двух типов компонентов. Рассмотрены недостатки и структура интерфейса к транспортной службе. Показана структура сделанной реализации предлагаемой транспортной службы.

MTA SZTAKI Tanulmányok 183/1986 pp 51-57

СОЗДАНИЕ МНОГОМАШИННОГО КОМПЛЕКСА НА БАЗЕ ОПЕРАЦИОННОЙ СИСТЕМЫ BOS 2

Заре Зарев, Александер Ахегукян, Валя Петрова
Елена Попова

Развитие вычислительной техники в последние годы создало условия для построения мощных вычислительных комплексов. Связывание вычислительных машин для работы в многомашинном комплексе дает пользователю много преимуществ. Он имеет доступ к значительно большими вычислительными ресурсами, чем ресурсы, предоставляемые его машиной. Благодаря коллективному использованию вычислительных ресурсов многомашинного комплекса, наличная вычислительная техника употребляется более эффективно.

В настоящей работе обсуждается многомашинный комплекс, созданный в Институте математики БАН, на основе МЭВМ СМ-4. Каждая машина работает под управлением операционной системы bos 2. Связь между машинами осуществляется при помощи некомутируемых линий. Для каждой машины связанные с ней другие машины представляют терминалы. Для связи с удаленной машиной используется устройство ИЗОТ 6003, которое представляет несколько терминальных интерфейсов, обьединенных в одно устройство. Если расстояние между машинами не более 600 метров, то используется токовая петля, в противном случае используются модемы. Связь является нискоскоростной и работа выполняется в режиме "старт-стоп".

Для функционирования сети необходимо сделать на каждой машине физическое и логическое описание конфигурации.

Физическое описание представляет описание связей между машинами и средств для их реализации.Для этого на каждой машине создается файл,который содержит описание непосредственно связанных с ней других машин.Каждой машине в рамках многомашинного комплекса присваивается уникальное имя,чем она идентифицируется на каждом сеансе.

Логическое описание представляет описание прав абонентов в сети.Для этого на каждой машине создается файл,который содержит информацию о правах потребителей комплекса,о способах реализации установления связи между машинами и т.д.

Обмен информацией между машинами осуществляется на основе пакетного протокола.Формат пакета не содержит информации об адресе получателя,т.е. работа выполняется без маршрутизации.Чтобы потребитель смог передать информацию от/к определенной машине многомашинного комплекса,он сам должен определить маршрут передаваемой информации.

Для работы в многомашинном комплексе каждый абонент комплекса имеет возможность пользоваться средствами системы UUCP (user to user copy).Система UUCP представляет пакет программ,осуществляющих передачу данных между машинами и дистанционное выполнение команд.Некоторые из этих программ,потребитель выполняет как стандартные команды операционной системы,указывая заданным командным языком требуемые обекты и параметры.Другие,демоны системы UUCP, активизируются системой.

Система предоставляет следующие команды:

- uucp - для копирования файлов с одной машины на другую
- uux - для выполнения команд на удаленной машине
- uulog- для обработки системного журнала.

Команды создают спомагательные файлы в рабочей директории, содержащие информацию об обмене и используемые для осуществления работы между машинами.Демоны системы UUCP осуществляют передачу в многомашинном комплексе,на основе обработки спомагательных файлов.Существует три типа спомагательных файлов:

- рабочие файлы - содержат директивы для передачи файлов между машинами
- файлы данных - содержат данные для передачи к удаленным машинам
- выполнимые файлы - являются директивами выполнения команд BOS2 с использованием ресурсов разных машин.

При помощи примерной сети из фигуры 1,проиллюстрируем выполнение команд системы UUCP в многомашинном комплексе.

Пример 1.

Потребителю машины m1 ,с именем user1 ,нужно копировать файл filea с машине m3 в файл filea на машине m4 .Надо выдать команду:

```
uucp  m3!filea  m4!filea
```

В машине m1 создается рабочий файл,которы посылается в машину m3.Там выполняется сгенерированная команда

```
uucp  filea  m4!filea
```

Копирование выполняется только если потребителю user1 разрешено копирование с m3 на m4 .

Пример 2.

Потребителю машины m5 нужно сравнить команды uucp в директориях

/usr/bin машины m3 и машины m4 .Надо выдать команду

```
uux "m4!diff m3!/usr/bin/uucp m4!/usr/bin/uucp m5! file1"
```

Для выполнения команды diff в машине m4 ,необходимо наличие файла usr/bin/uucp машины m3 в машине m4 .Файл usr/bin/uucp копируется с машины m3 на машину m4 .Если потреб ель имеет разрешение выполнять команду diff ,она выполняетс на машине m4 .Результат выполнения команды записывается под именем file1 в директорию потребителя в машине m5

Пример 3.
Надо сделать копирование всех файлов из директории /usr/lib машины m5 ,заканчивающие на .H ,на машину m1 .Нельзя выдавать команду

```
uux "m6!uucp m5!/usr/lib/*.H m1! xxx"
```

в машине m1 ,потому что между машинами m1 и m5 не существует директная связь.Посредством команды

```
uux "m6!uucp-d m5!/usr/lib/*.H m6!/usr/work"
```

осуществляется копирование файлов /usr/lib/x.H машины m5 в директорию /usr/work машины m6 /создание директории /usr/work в машине m6 гарантируется опцией -d /.Команда

```
uucp m6!/usr/work/ .H xxx
```

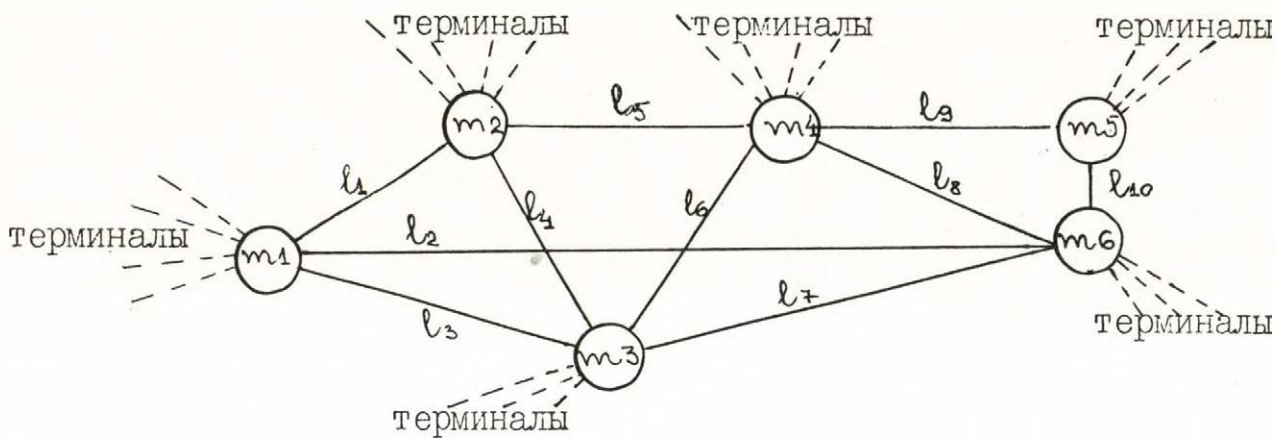
копирует требуемые файлы из m6 в директорию машины m1 /если нет абонента машины m1 с именем xxx /.Если среди абонентов машины m1 есть абонент с именем xxx,файлы записываются в его начать-

ной директории.Каждый абонент имеет доступ к всем ресурсам многомашинного комплекса.Особое значение при таком режиме работы,надо отдавать защите информации. Система UUCP,если ее оставить без ограничений,будеть позволять любому потребителю выполнять всякие команды и копировать к себе или от себя любой файл пользуясь прав вхождения в определенную машину комплекса.Кроме стандартных средств защиты системы BOS2,имеется в разпоряжении собственные средства защиты системы UUCP,которые устанавливаются при инсталлировании пакета.

Администрирование работы системы UUCP в многомашинном комплексе осуществляется при помощи стандартных возможностей операционной системы и посредством специальных средств программ и файлов ,которые упрощают поддержку и обеспечивают надежность работы.

Система работает со сравнительно большой надежностью.Связь осуществляется на принципе повторения передачи информации при сбоев.Если определенная машина работает нестабильно,то она блокируется на некоторое время,после чего сеанс связи повторяется Интервал времени блокировки устанавливается при инсталлировании программной системы UUCP.

В заключении надо отметить,что благодаря предоставленным средствам передачи данных с одной машины на другую и выполнения команд системы BOS2 на произвольной машине многомашинного комплекса,потребителям предоставляется удобное и мощное средство для работы с распределенными даннами.



Физическое описание примерной сети:

в машине m1
- m2 l_1
- m3 l_3
- m6 l_2

в машине m2
- m1 l_1
- m3 l_4
- m4 l_5

в машине m3
- m1 l_3
- m2 l_4
- m4 l_6
- m6 l_7

в машине m4
- m2 l_5
- m3 l_6
- m5 l_9
- m6 l_8

в машине m5
- m4 l_9
- m6 l_{10}

в машине m6
- m1 l_2
- m3 l_7
- m4 l_8
- m5 l_{10}

A MULTICOMPUTER CONFIGURATION BASED ON THE BOS2 OPERATING SYSTEM

Zare Zarev, Alexander Ahegukian, Valia Petrova
Elena Popova

ABSTRACT

The present paper discusses a multicomputer configuration designed at the Institute of Mathematics affiliated to the Bulgarian Academy of Sciences using CM-4 microcomputers. Every computer in the configuration runs under control of the BOS2 operating system. The separate units communicate by means of hard-wired communication lines.

The multicomputer configuration operates by help of the UUCP program system. UUCP is a series of programs employed for file transfers and remote command execution.

MTA SZTAKI Tanulmányok 183/1986 pp 59-67

О КОМАНДАХ BOS2

Христина Зашева, Десислава Димкова
Единый центр математики и механики, БАН

Команды – это основное средство, при помощи которого операционная система для МЭВМ BOS2 осуществляет взаимодействие с пользователями. Читателям, знакомым с операционными системами машин ЕС, будет легче понять суть простой команды, если ассоциируют это понятие с понятием "утилита". Иначе говоря, каждая простая команда – это объектный код, который отличается своим именем и находится в таком месте, которое известно системе.

Кроме того, BOS2 дает возможность из простых команд строить сложные командные структуры – канальные строки, списки и команды высокого уровня, так что пользователь с небольшим опытом может решать весьма трудные задачи. Конечно, для пунктов, в которых системные команды не обладают высокими качествами (ввиду своей универсальности) можно писать (на языке Ассемблера или на языках высокого уровня) свои простые команды, расширяя таким образом возможности системы и делая тем самым свою реализацию эффективней.

Простая команда – это последовательность слов, несодержащих пробелов и разделенных пробелами. Первое слово – имя команды, а остальные – ее аргументы, которые передаются внутри команды.

В зависимости от назначения и выполняемых действий, команды которые система предоставляет для общего пользования можно объединить в несколько групп:

- команды управления сессией
- команды работы с файлами
- команды обслуживания файловой системы

- команды, дающие средства для управления процессами, при помощи которых осуществляется выполнение команд
- команды поддержки коммуникаций в системе
- команды, дающие средства создания новых программ
- команды редактирования и обслуживания текстовых файлов
- другие команды.

Рассмотрим несколько подробнее возможности, которые команды каждой группы предоставляют потребителю операционной системы.

Управление сессией

К этой группе можно отнести команды для регистрации пользователя, для установления и изменения пароля пользователя, для установления опций терминала и получения его имени, для учета использованных машинных рессурсов.

Команда **login** - это первая команда, которая нужна пользователю, чтобы он начал работать в системе. При помощи единственного аргумента этой команды он указывает свой идентификатор абонента системы (UID), тем самым регистрируется и начинает сеанс.

```
login ivan
```

подключает в систему потребителя, идентификатор которого **ivan** . Команду можно выполнять в любое время для изменения идентификатора потребителя и перехода с одного пользователя к другому.

Возможность установления или изменения существующего уже пароля дает команда **passwd** . Каждый абонент, по желанию, может установить свой пароль. Если пароль установлен, система требует его каждый раз, когда абонент регистрируется в системе.

Команда **stty** устанавливает опции ввода/вывода на текущем выходном терминале. Задавая только имя команды без аргументов можно получить текущее состояние опций. При помощи аргументов можно:

- определять длину строки (в символах);
- определять длину экрана (в строках);
- установить режим стирания символа или отменить этот режим;
- замещать символы табуляции пробелами или отменить этот режим;
- изображать контрольные символы или отменить их изображение

и другие.

К той же группе относится и команда **su** , дающая возможность не изменяя окружение пользователя (в том числе текущую директорию) переключить на другой идентификатор пользователя.

Информацию о текущих пользователях **BOS2** можно получить выполняя команду **who** без аргументов. Эта информация включает для каждого регистрированного абонента: идентификатор, имя терминала и время регистрации. Команда с аргументами в виде **who am i** сообщает как вы зарегистрированы.

Имя терминала, за которым вы работаете можно получить выполняя команду **tty** .

Работа с файлами

К этой группе относим команды, дающие средства вывода, копирования, перемещения, сравнения, удаления файлов и изменения режима доступа к файлу.

Команды, распечатывающие содержимое файла или некоторой его

части - это **cat** , **lpr** , **pr** и **tail** . Для первых трех команд в виде аргументов можно задавать более одного имени файла.

```
pr file1 file2 file3
```

cat выводит их содержимое в указанной последовательности, без какой бы то ни было дополнительной информации и без всяких разграничителей между отдельными файлами. **lpr** осуществляет печать файлов в фоновом режиме, организуя очередь и выводя их содержимое когда устройство печати становится доступным. Файлы отличаются идентификатором пользователя, появляющимся на первой странице распечатки каждого файла. **pr** кроме содержимого выводит и информацию, включающую дату и час последней модификации файла, его имя и номер страницы. Существует возможность указать вывод в нескольких столбцах, как и параллельный вывод нескольких файлов.

При помощи аргументов команды **tail** можно распечатать часть файла, начиная с начала или с конца файла. При этом размер распечатываемой части можно указывать в строках, блоках или символах.

```
tail -20l xyz
```

означает что будут распечатаны последние 20 строк файла **xyz** .

Командами копирования (**cp** и **dd**) можно копировать файл в файл, или несколько файлов в директорию, как и воспользоваться некоторыми преимуществами поточного обмена.

Полезными командами этой группы являются еще команда сравнения двух файлов (**cmp**), команда переименования и перемещения существующих файлов (**mv**), команда удаления элементов одного или

больше файлов из директории (rm), как и удаления элементов целой директории (rmdir). Следует быть очень осторожными при выполнении последних двух команд.

Основной командой в этой группе является команда установления или изменения режима доступа к указанным файлам и директориям chmod . Режим можно задавать в абсолютном (восьмеричное число) или символьном виде и определяется для трех типов пользователей - для владельца (u), группы (g) и для других пользователей (o). Для каждого типа пользователей определяются разрешения на чтение (r), на запись (w) и на выполнение (x).

```
chmod ug=rwx file1
```

присваивает полные разрешения (на чтение, запись и выполнение) доступа к file1 для владельца и группы (u и g).

Обслуживание файловой системы

К этой группе относится большое число команд, но мы остановим свое внимание только на те, которые употребляются чаще всего и являются наиболее необходимыми при повседневной работе. Они обеспечивают возможности получения информации о состоянии носителя, поддержки файловой системы - восстановления, обновления управляющего блока, создания архива на магнитной ленте.

pwd , cd и ls связаны с работой пользователя в текущей директории - pwd выводит полное ее имя, cd меняет текущую директорию на указанную единственным аргументом команды, т.е. осуществляется переход и новая директория становится рабочей. ls без аргументов распечатывает содержимое текущей директории. Если

в качестве аргумента задано имя директории - команда выдает ее содержимое. При помощи т.н. "опций" можно получить список содержимого, упорядоченным по разным признакам - алфавитному, по времени последней модификации, по времени последнего доступа и т.д.

Новую директорию (или директории) можно создавать выполняя **mkdir** .

Справку о свободном дисковом пространстве (в блоках) выдает команда **df** , а при помощи **du** выдается справка, содержащая для всех файлов (или для указанного) число занимаемых блоков.

Проверку и восстановление файловой системы можно осуществить при помощи **icheck** . Вывод команды обычно содержит:

- общее число файлов и число обычных файлов, директорий, блочных специальных и символьных специальных файлов;
- общее число использованных блоков;
- число свободных блоков;
- число пропущенных блоков (тех, которые не входят ни в одном файле, ни в списке свободных блоков.

Команда записи и восстановления файлов на/с магнитной ленты - это команда **tar** . При помощи аргументов можно указать:

- извлечение файлов из ленты;
- добавление файлов на ленту;
- создание новой ленты;
- блокирование файлов при записи и др.

Выполнение команд

Выполнение команд и управление процессами реализуется средствами командного интерпретатора **shell** и команд **echo** , **kill** , **ps** , **sleep** , **test** , **wait** и др. Подробное описание функцио-

нирования этих команд можно найти в статьях "Управление процессами" и " SHELL командный интерпретатор".

Коммуникации в системе

В системе существуют развитые средства для поддержки коммуникаций с другими подключенными пользователями во время работы, как и средства электронной почты.

Средства посылать или получать почту дает **mail** . При этом можно менять порядок распечатки сообщений по принципу "первым вошел - первым вышел" (**first-in first-out**), как и для каждого сообщения указать свою реакцию - записать в т.н. "почтовый ящик", стереть, перейти на следующее, вернуться к предыдущему и т.д.

Команда **write** дает средства подключенным потребителям посылать друг другу сообщения во время работы за терминалом. Аналогичной по действию является **wall** , только она позволяет привилегированному потребителю посылать сообщения всем подключенным пользователям.

Программирование

Сюда относим команды, дающие средства написания новых программ. Программы можно писать на языках Ассемблер, С и Паскаль.

Основными командами этого раздела являются

as - вызов компилятора с языка Ассемблер;
cc - компилятор с языка С;
pi - вызов компилятора с языка Паскаль;
ld - редактор связей;
nm - выводит таблицу символических имен объектного модуля;
od - делает дамп заданного файла;

size - сообщает размер сегментов указанного объектного модуля;

tsort - дает возможность сделать топологическую сортировку и другие.

Описание действия этих команд можно найти в "Программирование в системе **BOS2** ".

Редактирование и обслуживание текстовых файлов

Эта группа обеспечивает средства для создания и поддержки текстовых файлов, в том числе для функций редактирования как включение, исключение, модифицирование, вывод строк. Допустимо использование кириллского и латинского алфавитов, как и прописных и строчных букв.

Основной командой является текстовой редактор **ed**, действие которого, как и некоторых других команд этой группы описаны в статье "Работа с текстовыми файлами".

Другие команды

В системе **BOS2** есть несколько команд, которые трудно отнести к одному или другому классу. Здесь можно упомянуть команду, распечатывающую текущую дату и час по машинным часам - **date** , пакет программ, дающий возможность арифметических вычислений с произвольной точностью, вызываемый командой **dc** , т.н. электронную записную книжку **calendar** , позволяющую получать напоминающую информацию на сегодня и завтра.

ON THE COMMANDS IN BOS2

Christina Zasheva, Dessislava Dimkova

ABSTRACT

Most of the programs available as BOS2 commands are listed. Depending on their functions and facilities commands are grouped as follows: user access control and terminal handling, file manipulation, system maintenance, running of programs, communications, program development tools, editing and others.

MTA SZTAKI Tanulmányok 183/1986 pp 69-77

ОБУЧАЮЩАЯ СИСТЕМА ПО ИСПОЛЬЗОВАНИЮ ОПЕРАЦИОННОЙ СИСТЕМЫ BOS2

Христина Зашева, Десислава Димкова
Единый центр математики и механики, БАН

Работа с некоторой операционной системой требует определенного минимума знаний об ее структуре, средствах и возможностях. Приобретение таких знаний часто оказывается неприятным, трудоемким процессом, включающим изучение документации довольно большого объема. При этом, такая документация содержит ненужную для начинающего потребителя информацию. Целью обучающей системы является облегчение процесса ознакомления потребителя с основными понятиями и возможностями операционной системы для МЭВМ BOS2 . Система LEARN - это пакет для самообучения, который сбережет потребителю усилий для накопления опыта в овладевании тонкостями работы, предоставляя ему готовые примеры использования некоторых наиболее употребляемых компонент системы.

ОСНОВНЫЕ ПОНЯТИЯ И ОБЩАЯ СХЕМА РАБОТЫ

Пакет состоит из основной программы LEARN , управляющей работой в курсах самообучения и из трех курсов, проводящих обучение на русском языке. Управляющая программа написана на основном языке системы BOS2 - языке C и выполняется как стандартная команда операционной системы, а курсы вызываются программой по желанию.

Курсы, которые реализованы в настоящей версии

EDITOR - подробно знакомит с возможностями текстового редактора - дает познания об основных функциях создания и редактирования текстов и файлов:

добавление, стирание и замена строк, записывание из одного файла в другой и др.;

FILES

MOREFILES - знакомят потребителя со структурой файловой системы и с работой с файлами - дают познания о расположении файлов в директории, о средствах распечатывания, переименования, копирования, нахождения, стирания, сравнения файлов и пр.

Каждый курс – это последовательность уроков, которые содержат объяснительный текст, примеры и вопрос или задание к обучающемуся. Уроки написаны на авторском языке системы и интерпретируются управляющей программой.

Каждому рассматриваемому в курсе объекту (это может быть команда, специальный знак, указатель и т.д.) посвящено несколько уроков (от 3 до 10–11) различной трудности. Урок начинается коротким объяснением о рассматриваемом объекте и примером. Потом от потребителя требуется повторить пример или составить его разновидность. Все предусмотрено быть настолько простым, чтобы большая часть обучающихся могла ответить правильно на большинство вопросов затверждающие знания.

Уроки составлены в трех вариантах. Простейший из них излагает урок и задает вопрос, которому надо ответить "yes" (да) или "no" (нет). Примером урока такого типа может быть:

Команда "ls" распечатает список имен файлов в вашей директории. Существует ли файл с именем "junk" ? Проверьте и потом ответьте "yes" , если существует или "no", если нет.

Прежде чем ответить, потребителю предоставляется возможность экспериментировать.

Уроки второго варианта требуют ответа в виде слова или числа. Прежде чем ответить, обучающийся должен выполнить несколько команд, результат которых и определяет его ответ. Урок о файлах, например, мог бы заканчивать вопросом:

Сколько файлов содержится в текущей директории?
Напишите "answer N" , где N - число файлов.

В уроках третьего варианта, который есть наиболее распространенный вариант, обучающийся должен выполнить задание в виде последовательности команд, осуществляя диалог с машиной. Закончив задачу, обучающийся должен написать "ready" .

После того как потребитель закончит введение ответов, начинается проверка их правильности. Если ответ верен, программа поздравляет обучающегося и пересылает его на следующий урок. В противном случае она отвечает соответным сообщением и дает обучающемуся возможность выбрать ввести заново ответ или перейти на новый урок.

За каждый правильный ответ обучающийся получает 1 очко, а за неправильный теряет 1 очка. Очка, накопленные обучающимся в течении одной сессии с программой LEARN заносятся в журнал программы и называются СКОРОСТНОЙ ХАРАКТЕРИСТИКОЙ.

LEARN все время следит за скоростной характеристикой, которая сильно влияет на работу с ней. В зависимости от нее определяется путь, по которому обучающийся проходит уроки. Таких основных путей (дорожек) - три.

Тот, у которого большая скоростная характеристика (около 10),

движется по кратчайшей дорожке. В большинстве случаев, двигаясь по этой дорожке, обучающийся проходит один только урок из группы уроков, посвященных данному предмету и это самый трудный урок в группе. Если ответ правилен, обучающийся остается на кратчайшей дорожке. В случае ошибочного ответа, отняв 4 очка от скоростной характеристики, система сама ищет уроки того самого предмета, но более легкие, т.е. движение продолжается по некоторой уже более длинной дорожке.

Когда скоростная характеристика не очень велика (около 5), учебный материал распределен в нескольких (обычно 2-4) уроках. Если в результате успешного выполнения заданий потребитель накопит необходимое для кратчайшей дорожки число очков, он автоматически перебрасывается на нее.

Самый длинный путь обучающийся проходит, когда у него очень маленькая скоростная характеристика (0-2 очка). Как и в прежнем случае, если в результате хорошей работы у него накопится достаточно очков, он автоматически перебрасывается на более короткую дорожку.

Дорожки, о которых идет речь не являются вполне независимыми, наоборот, существуют уроки, через которые все три проходят.

В течении одной сессии в программе LEARN невозможно два раза встретить один и тот же урок. Иными словами, нет путей с самопересечениями.

ФАЙЛОВАЯ СТРУКТУРА

Фаловая структура программы LEARN выглядит следующим образом:

```
lib
   play
       st1
          files
       st2
          files
   files
       L0.1a
       L0.1b
       ----
   editor
       L0.1a
       L0.1b
       ----
   morefiles
       L0.1a
       L0.1b
       ----
   log
```

Существует производящая директория lib , в которой сохраняются все данные о курсах.

В директории lib находятся следующие поддиректории:

files, morefiles, editor - содержат уроки трех курсов;

log - журнал регистрации результатов обучающихся;

play - здесь создаются потребительские поддиректории.

Каждому потребителю, работающему с программой LEARN создается частная директория в файловой системе. Эта директория содержит файлы, с которыми обучающийся будет работать и по окончание сессии стирается.

АВТОРСКИЙ ЯЗЫК ПРОГРАММЫ

Авторский язык программы LEARN состоит из команд, которые задаются в виде строк, начинающихся символом "#".

#Print

- эта команда распечатывает заданный текст, начиная строкой, следующей за командой и, встретив снова символ # , останавливается.

#Print file

- распечатывает содержание файла file .

Если обучающийся не ответил правильно и хочет повторить задание, команда #Print , в обоих ее формах, не распечатывает повторно вводный текст.

#create filename

- создает файл с заданным именем и записывает в нем следующие строки, до встречи символа # . Эта команда используется для создания рабочих файлов и подачи справочных данных для урока.

#user

- эта команда передает управление обучающемуся и посылает каждую написанную им строку командному интерпретатору для исполнения. Действие команды #user кончается когда обучающийся напишет Yes , no , answer или ready .

#copyin

#uncopyin

- все, что обучающийся запишет между этими двумя командами составляет содержание специального файла .copy и это дает программе возможность истолковать ответы на поставленные вопросы (задания).

#copyout

#uncopyout

- материал, заданный обучающемуся и включенный между этими двумя командами составляет содержание специального файла .ocopy и это дает программе возможность оценивать эффект команд обучающегося.

#pipe

#unpipe

- обычно, команды обучающегося передаются для исполнения командному интерпретатору shell построчно. В случаях, когда данная последовательность строк предназначена для некоторой программы как ее стандартный ввод, она вставляется между командами #Pipe и #unpipe и посылается непрерывно по каналу. Чаще всего это используется для ввода подкоманд построчного редактора. Когда предусмотрено использование и пары команд copyout , то она обязательно включает в себе пару PiPe .

Перечисленные ниже команды определяют состояние после введения ответов.

#cmP file1 file2

- сравняет идентичность обоих файлов.

#match stuff

- сопоставляет последнюю строку обучающегося со stuff , от чего зависит состояние успеха или неуспеха. Все дополнительное, как слово answer например, удаляется еще до сопоставления. Если возможно дать несколько правильных ответов, в уроке может существовать несколько строк, начинающиеся командой #match .

#bad stuff

- эта команда совсем похожа на команду #match , но используется в случаях, когда составители урока предусматривают некоторые специфические ошибочные ответы.

#succeed

#fail

- выдает сообщение об успехе или неуспехе.

Кроме вышеуказанных команд состояние оценивается еще и при помощи обычных команд BOS2 . Типичные измежду них - GreP и test . Эти команды выдают сообщение о состоянии: верно (0), если задача решена верно и неверно (отличное от нуля) в противном случае.

#log file

- записывает в файл file день, час, урок, имя обучающегося, скоростную характеристику и индикацию об успехе/неуспехе.

#log

- записывает вышеупомянутую информацию в журнал программы LEARN .

#next

- последованное несколькими строками типа

  25.1a 10
  25.2a 5
  25.3a 2

определяет номер следующего урока в зависимости от скоростной характеристики.

COMPUTER-AIDED INSTRUCTION ON BOS2

Christina Zasheva, Dessislava Dimkova

ABSTRACT

A program for interpreting CAI scripts on the BOS2 operating system for minicomputers and a set of scripts providing a computerized introduction to the system are presented. The file structure used by the program and the command language for writing scripts are described.

MTA SZTAKI Tanulmányok 183/1986 pp 79-84

СИСТЕМА ДЛЯ ТЕКСТООБРАБОТКИ В ОПЕРАЦИОННОЙ СИСТЕМЕ БОС2

И. Денев, Е. Живкова, М. Филипова
Единый центр математики и механики.
Болгарская Академия Наук

В ЕЦММ, Болгарской Академии Наук коллективом сотрудников из НПЛ "Программа" разработана операционная система БОС2 для мини ЭВМ типа СМ-4. Прототипом этой системы является широко известная система *UNIX*, версия 7.

БОС2 включает в себя весьма мощный пакет программ для текстообработки. Основными компонентами этого пакета являются:

- базовая текстообрабатывающая программа;
- универсальная макробиблиотека;
- программа для форматирования таблиц.

В пакете включены и некоторые другие программы, выполняющие вспомагательные функции, которые связаны с возможностями устройства вывода.

Ядро пакета - это базовая текстообрабатывающая программа. По существу это не текстообработка, а язык для создания разнообразных форматирующих систем, ориентированных к потребностям пользователей. Для этого используется механизм макроопределений. Язык базовой текстообрабатывающей программы предоставляет практически неограниченные возможности для манипулирования текстами, в число которых входят:

- управление форматом страниц - выбор шрифта, определение длины строки и междустрочных интервалов, выравнивание текста, в том числе автоматический перенос слов, задание длины страниц, многоколонный вывод, и т.д.;

- управление процессом обработки, при помощи условных операторов, переключений ввода/вывода, программных прерываний и др.;
- применение внутренних числовых и символьных данных - числовые регистры и цепочки;
- средства для описания макроопределений;
- большое число специфических средств - табуляция, заголовки, нумерация страниц, строк и др.

Вводной текст для текстообрабатывающей программы состоит из:
- текста, подлежащего форматированию и выводу;
- командных строк;
- управляющих последовательностей в форматируемом тексте.

Общее число команд базовой текстообрабатывающей программы превышает 80, что ограничивает их детальное рассмотрение в настоящей работе.

Необходимо отметить, что базовые средства текстообработки реализуют очень детальный язык. Его применение для манипулирования текстами предполагает высокую квалификацию пользователя в области электронной обработки информации. Для того, чтобы эти средства стали доступными потребителям разных профилей, разработан язык высокого уровня. Его можно легко расширять и модифицировать в зависимости от класса пользователей.

Он является более удобным для составления и обработки документов разных типов, как статьи, отчеты и т.п., по следующим причинам.

Язык отражает структуру документов с пользовательской точки зрения. Он работает с такими единицами как секция, параграф, таблица, фигура, подстрочное примечание, содержание документа и т.п., хорошо известные и до появления средств текстообработки при помощи ЭВМ.

С другой стороны, через него осуществляется доступ к программным средствам для автоматизации функций по составлению и обработ-

ке документов, как:

- Разбивание текста всего документа на страницы и его вывод в разных форматах и на разные устройства.

- Нумерация страниц и снабжение каждой выводимой страницы документа со заголовными и/или подписывающими строками, как и управление их размещением на странице и относительно ее содержания.

- Оформление части текста в виде параграфа. Применение некоторых операторов языка облегчает пользователя при составлении разных типов параграфов и при их дальнейших изменениях.

- Снабжение секций текста номерами и заголовками, при чем номер генерируется автоматическим способом. Операторы, реализующие эту функцию, освобождают пользователя от поддержки единой нумерации секций при включении новых или при удалении уже существующих.

- Управление расположением части текста в рамках всего текста. Ряд операторов предоставляют возможности для вывода таких частей текста, как списки, фигуры, цитаты, таблицы, при чем произходит автоматическое форматирование и расположение текста.

- Перенаправление вывода части текста. При помощи некоторых операторов пользователь может указать части входного текста для сохранения с целью дальнейшего вывода. Он произходит по разному - в некоторых случаях он определен заранее и осуществляется автоматическим способом, как например вывод подстрочных примечаний; в других - пользователь должен указать сам место вывода, как при выводе элементов индексов, содержания и т.п.

Применение языковых и программных средств освобождает пользователя от весьма неприятных деятельностей после корректировки текста документа.

Язык разработан на основе языка базовой обрабатывающей программы. Он состоит из макрооператоров, при чем их определения включены в некотором стандартном файле, названном еще универсальной макробиблиотекой. На ее основе можно получать новые макробиблиотеки, в зависимости от пользовательского профиля, для которого они предназначены.

Для оформления таблиц с помощью только средств базовой текстообработки необходимо выполнить ручным способом довольно трудоемкие операции - вычисление ширины столбцов , выравнивание , оформление заголовков и т.д. Для автоматизации этого процесса разработана программа, которая позволяет легко описывать довольно сложные таблицы. Описание каждой таблицы независимо от остальных и содержит форматирующую информацию и данные , составляющие таблицу. Формат описывает отдельные столбцы и строки в таблице. Кроме него в описании таблицы могут присутствовать опции , которые действуют на таблицу в целом.

ОПЦИИ. Через опции можно запросить центрирование таблицы относительно текущей длины строки , расширение таблицы до текущей длины строки , или задать явно символ табуляции. Действие указанных опций распространяется на всю таблицу.

ФОРМАТ. Форматная секция таблицы задает макет каждого столбца таблицы. Каждая строка в этой секции соответствует одной строке таблицы. Исключением составляет последняя строка в форматной секции, которая соответствует всем последующим строкам таблицы. Форматная строка содержит ключевые символы для каждого столбца таблицы. С помощью ключевого символа можно задать левое, правое выравнивание, или центрирование столбца; определить столбец с числовым выравниванием, при котором отчитывается положение десятичной точки числа; определить алфавитный подстолбец, который выравнивается слева и располагается так, что самый длинный его

элемент центрирован в столбце; или определить соединенное заглавие.

Кроме основных ключевых символов предоставляется и ряд дополнительных возможностей, которые задаются после основного ключевого символа. Можно задать минимальную ширину столбца; расстояние, разделяющее описываемого столбца от следующего. Также можно запросить одинаковую ширину нескольких столбцов.

ДАННЫЕ. Данные для таблицы задаются после форматной секции. Обычно, каждая табличная строка задается как одна строка в секции данных. Но длинные вводные строки данных можно разбить на несколько физических строк. Каждая вводная строка, которая заканчивает символом "\", комбинируется со следующей строкой и обе строки рассматриваются как одну. Данные для отдельных столбцов /элементы таблицы/ разделяются через символ табуляции.

Имеется возможность просто указывать вывод горизонтальных последовательностей в таблице:

- горизонтальную линию, длина которой равна ширине таблицы;
- горизонтальную линию, длина которой равна ширине описываемого столбца;
- цепочку из повторений указанного символа, длина которой равна ширине столбца.

Для форматирования более сложных таблиц, содержащих подзаглавий или суммирующих строк, предоставлена возможность после вывода части таблицы в определенном формате, изменить формат остальной части таблицы. При этом переходе от одного формата к другому не разрешается изменять число столбцов, расстояние между столбцами, выбор столбцов с одинаковой шириной, а также глобальные опции.

Программа для форматирования таблиц работает как предпроцессор к базовой текстообрабатывающей программе. Для этого очень удобен механизм канала в системе БОС2.

ЛИТЕРАТУРА

1. Беляков М.И. и др. Инстументальная мобильная операционная система ИНМОС. М.: Финансы и статистика, 1985.

2. Kernighan B.W. and Ritchie D.M. The C Programming Language. Englewood Clifs, NJ, Prentice-Hall,1978.

3. Kernighan B.W. and Pike R. The UNIX Programming Environment. Englewood Cliffs, NJ, Prentice-Hall.

DOCUMENT FORMATTING FACILITIES IN BOS2 OPERATING SYSTEM

J.Denev, E.Jivkova, M.Filipova

Abstract

BOS2 document formatting facilities are presented. The text-formatting program allows the user full control over right margin justification, automatic hyphenation, page titling and numbering, and so on. It also provides macros, arithmetic variables and operations, and conditional testing. There is a macro package that defines formatting rules and operations for a wide range of document preparation. There is a program to format tables providing an easy to learn language for producing tables of arbitrary complexity.

MTA SZTAKI Tanulmányok 183/1986 pp 85-94

A DOCUMENT-BASED INTERACTION MODEL

V. SABEV

Institute of Mathematics, Sofia

INTRODUCTION

The classic ways of interaction in the office are realized by personal talks, phone calls and exchanges of documents. Contrary to talks and calls, the third - document-based - way is slower. It does not need coordination and synchronization, and permits to store the information - temporary or permanently. Furthermore in practice there are sets of rules describing it. The exchange of documents is accomplished as a result. To some extent the presence of the rules allows some formalism of the third way of interaction.

The exchange of documents is computerized activ. For the computer-oriented automation of the document-based interaction exists already developed and are developed methods and technologies. But for the total automation of document handling it is necessary to computerize both the execution of the single operations and their initiation.

The subject of discussion is a model of the document-based interaction in the office. The emphasis is on the management of the operations of the third way of interaction, i.e. on their activating.

A FORMALISM OF THE DOCUMENT-BASED INTERACTION

Following the core of this way of interaction in the office it is clear that there are two kinds of objects to be formalized.

The first is the document, and the other - the sets of rules. In what follows the terms " administrative procedure " and " office procedure " are used for these sets.

There are some approaches and methods for formalized description and representation of documents and office procedures. The formalism discussed here is clasified to the class in which documents flow is emphasized. This class is refered as a class of precedence-oriented models (ElNu79). These models are named chronologic here. The precedence usually specifies control flow (Elli79) or it may show data flow (Hamm77). The models that are given in (Tsic82) belong to this class as well. They are based on the relatoionships between certain conditions dependent on the content of the documents and actions (operations) over the documents. In principle, the parallelism if it exists is at the level of office procedures in both models. They are simplified by some restrictions as multiple copies, synchronization, etc. An other approach belonging to this class is the information control net (Elli79). It includes explicitly the parallelism and synchtonization responsible to it. But this inclusion is valid for the level of operations, for different documents however. In some respects the following formalism sums up the results from the chronologic approaches and methods. At the same time the bases of the parallelism are developed in this formalism, the parallelism at the level of the operations concerning one document however. Actually, an effort is made to use the parallelism internal for the document and coming from it.

A). Formal Definition of Document

One document can be defined formally as 4-tuple :

$$d = (id, e, Str, v(t)),$$

where:

– id denotes the unique identifier of the document. id can be treated as a whole number, i.e. id belongs to N.

– e denotes the essence of the document. The generic–specific relations among the documents divide the whole domain of documents into subdomains (domains of financial, administrative, and other kinds of documents; letters, orders, reports , and so on). Let E be the finite nonempty set of identifiers used to denote each one of these domains. Then it can be treated that e belongs to E.

– Str denotes the structure of the document. It defines the inner organization of the document. Str can be an atomic structure or composed of some atomic or(and) of other composed structures according to certain rules for formation. Let STR be the nonempty set of structures. Then it can be treated that Str belongs to STR.

– v(t) is the current value of the document for the time t. v belongs to dom(Str), where dom(Str) is the domain of values of Str.

B). Structured Operation over the Documents.

The document is a structured individual with changeable values. Its structured character allows to search for a internal parallelism in it. This kind of parallelism needs two special operations, which in principle are to be independent on the document – its structure, its current value and so on. One operation is for the beginning of operations executed in parallel, i.e. as a result of it, there are images of the document - one image for each parallel operation. The other is for the termination of a parallel processing, i.e. from all the different images of a document one document is obtained. These operations are named structured.

Further it is assumed that there are no multi-valued components

in the structure of the documents. If there is such one, it will be treated as an ordered list of single-valued components. This restriction is important only for the structured operations. Moreover it is not obligatory for each component to have a "value". Each domain dom(X), where X belongs to STR, possesses one special value θ. It denotes the the empty value, i.e. the component having this value is "empty".

The first structured operation duplicates a document. Let d belong to D, and Dup denote this operation. Here D is the whole set of documents.. Then $Dup : D \rightarrow \{(d,d) \mid d \in D\}$. The so defined Dup is a total operation in D, i.e. it is valid for each d D.

The other structured operation reduces two images into one. Let Red denote this operation. Then $Let : D' \times D' \rightarrow D$, where $D' \subset D$. In contrast to Dup, Red is not a total operation. Further, one concrete form of Red is shown. For the definition of the concrete form a help function red will be used. For each Str belonging to STR and v_1 and v_2 of dom(Str), the function red is defined as follows:

(1). in the case of an atomic Str, the sets J and W are defined as well. $J = \{j \mid v_j \neq \theta,\ j=1,2\}$ and $W = \{v_j \mid j \in J\}$. In this case

$$red(v_1, v_2) = \begin{cases} \theta, & \text{if } J = \emptyset, \\ w, & \text{if } J \neq \emptyset \text{ and } |W| = 1, \text{ where } w \in W, \\ \text{indermined in other cases.} \end{cases}$$

(2). in the case of a composed Str, where $Str=(Str_1, Str_2, \ldots, Str_n)$ and $v_i=(v_{i1}, v_{i2}, \ldots, v_{in})$, $i=1,2$ and $n>1$,

$$red(v_1, v_2) = red((v_{11}, v_{12}, \ldots, v_{1n}), (v_{21}, v_{22}, \ldots, v_{2n})) =$$

$$= (red(v_{11}, v_{21}), red(v_{21}, v_{22}), \ldots, red(v_{1n}, v_{2n})), \text{ iff each}$$

$red(v_{1j}, v_{2j})$, $j=1,n$, is not indetermined. In other case red is indetermined as well.

Now the Red operation can be defined. Let $d_1=(id,e,Str,v_1(t))$ and $d_2=(id,e,Str,v_2(t))$ belong to D. Then

$$Red(d_1, d_2) = \begin{cases} d=(id,e,Str,red(v_1(t),v_2(t)), \text{ if red is determined,} \\ \text{indetermined in other cases.} \end{cases}$$

C). Formal Definition of Office Procedures

Following the core of the document-based interaction in the office it is seen that the architecture of the office procedures consists of a finite set of conditions, of a finite nonempty set of operations , and of a finite nonempty set of executors of the operations.

- condition. Each condition is a function of time, the essence, and the value of dom'(str), where str $\in$ STR, and dom'(str) $\in$ dom(str), and its values are 0 and 1, i.e. $C : R^+ \times E \times dom'(str) \longrightarrow \{0,1\}$ .Here C denotes a condition. The values of C mean dissatisfaction and satisfaction correspondingly. The rules of Boolean algebra hold for these values. Let at moment t a document d=(id,e,Str,v(t)) be given, and str be a "projection" of Str, and the projection $v_{str}(t)$ of v(t) corresponding to str belong to dom'(str). The C condition is dissatisfied at the moment t for the given document d, if $C(t,d_e,d_{v_{str}(t)}) = 0$, and in other case C is satisfied.

- situation. Let Con be a finite set of conditions, i.e. $Con = \{C_1, C_2, \ldots, C_n\}$,$n>0$, The situation is:

a). each condition or its negation, i.e. the items of $Con \cup \{\sim C \mid C \in Con\}$.

b). the elementary conjunction of sitiations if it is not identical equal to 0.

Let Sit denote the set of situations, and S be an item of Sit. S is satisfied at the moment t' for a document d, if each its condition and negation of condition are satisfied. It will be assumed that the "empty" situation, i.e. the situation, that does not consist of condition or negation, is identical satisfied.

- operation. Each operation is a function as follows: $A : dom(str') \longrightarrow dom(str")$, where A denotes an operation, and str' and str" belong to STR. Let at moment t a document d be given. After the operation A, by that d is handled, d is $(id, e, Str, A(v_{str'}(t)))$.

- executor. Each operation is realized by an executor. Let $Exe = Exe_1 \cup Exe_2$ be a finite nonempty set of executors. Here Exe_1 and Exe_2 are the sets of automated and manual ones correspondingly. It is clear that $Exe_1 \cap Exe_2 = \emptyset$.

- action. Let $Act = \{A_1, A_2, \ldots, A_m\}$ and $Exe = \{P_1, P_2, \ldots, P_q\}$ be finite nonempty sets of operations and executors correspondingly. The items of $PA = \{(P,B) | P \in Exe, B \in Act \times Act\}$ are named actions. Let $pa = (P_i, A_{j_1} A_{j_2})$ belong to PA. It means that the operations A_{j_1} and A_{j_2} are realized serial by the executor P_i. In this model it is assumed that the operation "identity" belongs to Act. In this way A_{j_2} can be the identity.

- direction. The direction is:

a). each action, i.e. the items of PA.

b). concatenation of directions.

The execution order of actions belonging to a direction is serial as well. The comming action is to be realized after the termination of the previous action of the direction, i.e. after the termination of all operations of the previous one.

- correspondence. Let Sit and Dir be finite nonempty sets of

situations and directions correspondingly. The relation between Sit and Dir is defined by the correspondence G ,i.e. $G \subseteq Sit \times Dir$. Let at the moment t the document d, the situation S, and the direction H be given. Then the situation S fixes the direction H for the document d at the moment t, iff (1) $S(t,d_e,d_{v_{str}})=1$ and (2) $H \in G(S)$.

Now an elementary office procedure for a document-based interaction in the office can be defined as 7-tuple:

$$EAP = (Con,Sit,Exe,Act,Dir,Dir_\omega,G)$$

where: (1) Con is a finite set of conditions; (2) Sit is a finite nonempty set of situations over Con and the negation of the items of Con (in the case of an empty Con, Sit consists only of the empty situation); (3) Act is a finite nonempty set of operations; (4) Exe is a finite nonempty set of executors; (5) Dir is a finite nonempty set of directions over the actions Exe x Act x Act, and $Dir_\omega \subseteq Dir$ is a set of "stop"-directions; (6) G is a correspondence, i.e. $G \subseteq Sit \times Dir$.

To use the parallelism, it is introduced here a next kind of office procedures - the hierarchical office procedure. This kind of procedures is similar to the kind of elementary ones. They differ from operations and executors. The operations and the executors of the hierarchical office procedures are terminal and nonterminal. The terminal ones are the same of the elementary office procedures. The nonterminal operations are other (elementary and hierarchical) office procedures. It is introduced nonterminal executors for the nonterminal operations. Let the action (M,A_1A_2) be given, where A_1 and A_2 are nonterminal operations, and M is their nonterminal executor. This action means the parallel execution of A_1 and A_2 by M. The handling of a document by such action consists

of 3 steps: (1) duplicating the document for each nonterminal operation of the action; (2) parallel execution of the nonterminal operations for the corresponding duplicated images of the document; (3) reducing of the duplicated images after the termination of all monterminal operations. This parallel execution is valid in the case of (1) independent nonterminal operations of one action and (2) the determined Red operation executed at the 3 step.

The hierarchical office procedure for a document-based interaction in the office can be defined as 7-tuple:

$$HOP = (Con, Sit, Exe, Act, Dir, Dir_{\omega}, G)$$

where: (1) Con is a finite set of conditions; (2) Sit is a finite nonempty set of situations over Con and the negations of the items of Com (in the case of an empty Con, Sit consists only of the empty situation); (3) Exe is a finite nonempty set of executors, where $Exe = Exe' \cup Exe''$, $Exe' \neq \emptyset$ is a set of terminal executors, Exe" is a set of nonterminal executors, and $Exe' \cap Exe'' = \emptyset$; (4) Act is a finite nonempty set of operations, where $Act = Act' \cup Act''$, $Act' \cap Act'' = \emptyset$, $Act' \neq \emptyset$ is a set of terminal ones, and Act" is a set of nonterminal ones; (5) Dir is a finite nonempty set of directions over the actions of $\{(P,B) \mid P \in Exe', B \in Act' \times Act'\} \cup \{(M,B) \mid M \in Exe'', B \in Act'' \times Act''\}$,and $Dir_{\omega} \subseteq Dir$ is a set of "stop"-directions; (6) G is the correspondence, i.e. $G \subseteq Sit \times Dir$.

CONCLUSION

The developed formalism extents the current models and approaches by using the internal parallelism into documents and by parallel-serial mode of their handling. The existence of the independent monterminal operations is a necessary condition, it is not sufficient one for the parallel-serial handling. The sufficiency needs operations

of kind of duplicate and reduce. As it is shown in the structured operation " reduce " it is not possible to realize all case of non-serial handling. The restriction depends on the reducing.

This model of office procedures is implemented by the determination of office procedure properties as completeness, noncontraversy, single- and multi-value. Morever, in order to be able to use the developed formalism it should be represented in computer-oriented form. For the computer-oriented form are used the decision tables adjusted and modified in accordance with the model of the hierarchical office procedures. In this way the computer-oriented representation of the hierarchical office procedure is already used both for analyzing the office procedure properties, and for a tabular language in some applications.

REFERENCES

Hamm77 Hammer M. etc.„A very high level programming language for data processing applications, CACM 20,11(Nov 77)

Elli79 Ellis C. A., Information control flow: a mathematical model of information flow, ACM proc. of the conf. on simulation, modeling, and measurement of computer systems, Boulder, CO, August 1979.

ElNu79 Ellis C. A. and G. Nutt, On the equivalence of office models, Xerox PARC, SSL-79-8.

Tsic82 Tsichritzis D.„Form management, CACM 25, 7(July 1982).

МОДЕЛЬ УЧРЕЖДЕНСКОГО ВЗАИМОДЕЙСТВИЯ НА ОСНОВЕ ДОКУМЕНТОВ

ВАЛЕНТИН СЪБЕВ
Институт математики, София

Объектом дискусии является модель учрежденского взаимодействия на основе документов. Ударение ставится на управление операциями этого взаимодействия, т.е. на их стартирования. В модели рассматриваются тоже и документы.

MTA SZTAKI Tanulmányok 183/1986 pp 95-101

CONCEPTUAL MODEL FOR OFFICE INFORMATION SYSTEMS

M. MARGARITOV

Central Institute of Computer Technics - Sofia
Bulgaria

INTRODUCTION

The Office Information System (IOS) is used for automation of office work.It is a complex system with unified user interface integrating many hardware and software components [1]. In recent years the research is directed to the design of the so called active OIS. Besides opportunities for automation and assistance of the basic office activities, they also allow automatic invocation, control and support of office procedures. The basic problem in the design of active OIS is the control module of the system. The paper presents a conceptual model for describing, modeling and analyzing of the conventional office environment and OIS. As a result a concrete model for the office to be automated is created. It is used for the design of the control module of OIS. The presented conceptual model provides means for description and modeling of structured and semi-structured activities.

1. INFORMAL MODEL DEFINITION

There is direct correspondence between the elements of the conceptual model (from now on called simply model) and the office components. The office procedure is the basic concept in model design. The procedure is defined as a set of activities which execution leads to predefined goals - postconditions of the procedure. The procedure invocation is caused when certain preconditions are met. Each procedure execution with a certain set of information objects is called a task. Activities are the building elements of the office procedure and manipulate with information

objects and resources. The information objects represent the logic and semantic structures of the information entities in office (documents, letters, etc). The information resource is a set of information objects gathered according to certain attributes (folders, archives, etc.). The information resources reflect the distribution of information objects in the office. Each office procedure or a part of it is executed by office workers or/and support devices and systems called agents. The ability of given agents to perform a set of activities is called a role.

2. FORMAL DEFINITION OF STATIC MODEL

As mentioned above, the office environment can be modeled as a set of procedures. The procedure can be represented as follows:

$$P = (A,\ IR,\ IO,\ C,\ R,\ W,\ \alpha,\ \beta,\ \delta,\ \sigma) \qquad (1)$$

where:

A is a finite nonempty set of activities in the procedure

IR is a finite nonempty set of information resources used in the procedure

IO is a finite nonempty set of information objects in the procedure

$C = C_{pre} \cup C_{post}$

C_{pre} is a finite nonempty set of procedure preconditions

C_{post} is a finite nonempty set of procedure postconditions

R is a finite nonempty set of roles in the procedure

W is a finite nonempty set of agents in the office

$\alpha: A \to A$ is a mapping of elements from set A to the same set, describing the order in which activities are performed

$\beta: A \to IR$ is a mapping of elements from set A to the set IR, defining the information resources used by activities

δ: A → IO is a mapping of elements from set A to the set IO, defining the information objects used by activities

σ: R → W is a mapping of elements from set R to the set W, defining the agents which perform the activity

3. DYNAMIC MODEL OF OFFICE PROCEDURE

Net model based on E-Nets [2] can be used to describe the dynamic characteristics of the office. A single net is assigned to every office procedure. Places and transitions are the basic net model elements. Transitions correspond to the activities in the procedure. The different types of activities correspond to different subsets of transitions. These transitions are more complex and generalized in comparison with the E-Nets or Petri-Nets [3]. This leads to significant simplification of the structure of the net model. Transition firing is performed in four phases in a way similar as in E-Nets. Places are viewed as attributes of the activities. Request arrivals to the activities are modeled by tokens arriving at the places. When a task is initiated a token is generated with an unique identifier. The marking of the net defines the distribution of requests in the places.

In this way the dynamic model of the procedure can be represented as follows:

$$P_d = (P, \mu)$$

where:

P is the static model of the procedure as defined in (1)

μ is the marking of the net corresponding to the procedure.

4. MODEL ELEMENTS REPRESENTATION

As earlier discussed, the basic model elements are: activities, information objects, information resources and agents.

The nature of the elements can be defined according to their properties - element attributes. The relations between every separate element with the others are also represented by its attributes. Elements with equal attributes are gathered in sets called element types. The element structure is reflected through hierarchical corelation between the attributes. At a given moment, every different element is defined by its attributes values.

Thus, every model element can be represented as a triple:

$E (id, T, V_t)$

where: id is unique element identifier

T is element type

V_t is element value in the moment t.

Element value (V_t) is defined as a vector of its attribute values at the moment t.

The proposed approach permits unified, internal representation of all model elements and the relationships between them.

5. MODEL APPLICATION

The presented conceptual model of the office is an universal tool for designing of the control module of active OIS (see Fig.1) The procedure, information object and agent libraries are based on the descriptions of office procedures, information objects and resources, agents and their roles. The Monitor controls the overall performance. It consists of Supervisor, Interpreter and Scheduller. The Supervisor checks the procedure library for activities which are ready to be executed and reports for them to the Interpreter. The activity execution is controlled by the Interpreter.

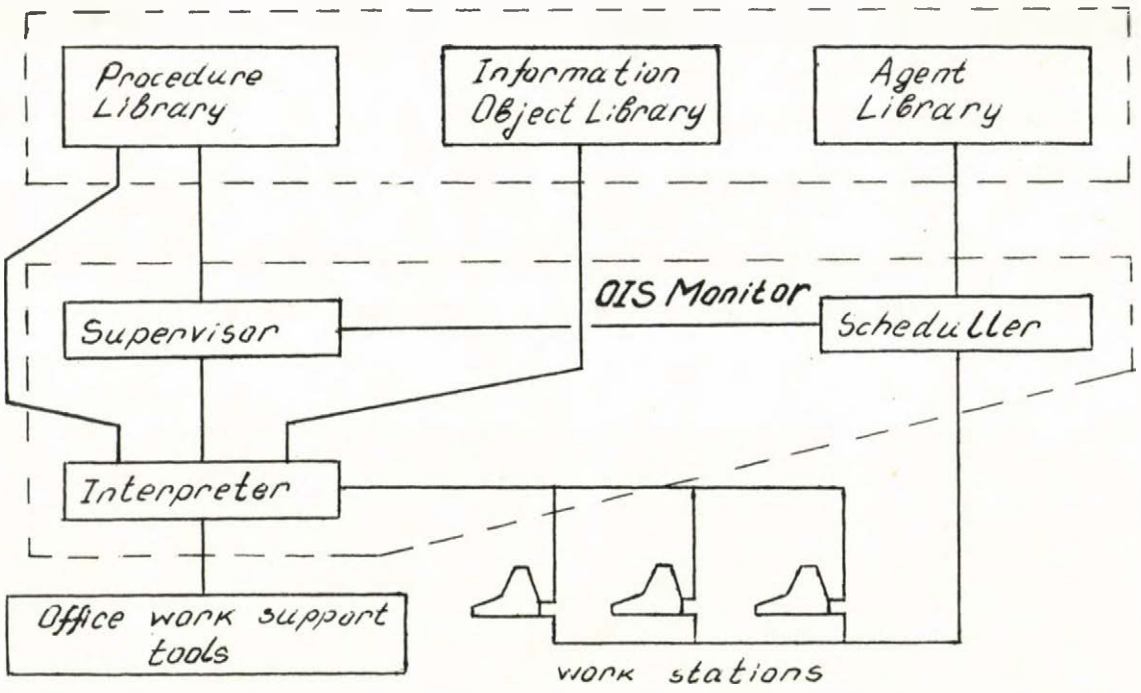


Fig. 1: Control module of active OIS

which communicates with the office support tools (electronic mail, textprocessing system, expert system, etc.) and the work stations. The Scheduller checks the status of the agents and allocates the activities among them.

CONCLUSION

The proposed model provides means for describing the procedural nature of information processing in the office and the information entities and agents in them. It can be used for modeling of parallel processing such as pipeline execution of tasks by single procedure, parallel processing of different procedure branches, concurent execution of different procedures. The

activities duration can be represented too. The model of the separate office has a modular structure and can be further detailed or generalized if needed which simplifies the design of the control module of active OIS.

REFERENCE

1. Янев, Кр. и Маргаритов, М., "Състояние на системите за автоматизация на административно-управленческата дейност", Автоматика, изчислителна техника и автоматизирани системи, бр. 10, 1985 г.

2. Nutt, G.J., "Evaluation nets for computer systems performance analysis", Fall Joint Computer Conference 1972, pp. 279-286.

3. Питерсон, Дж., "Теория сетей Петри и моделирование систем", Москва, "Мир", 1984.

Концептуальная модель автоматизированных систем учрежденческой деятельности

Михаил Маргаритов Маргаритов
Центральный институт вычислительной техники
1113 София, бул. Ленин, 7 км

В материале представлена концептуальная модель, предназначенная для описания, моделирования и анализа классической учрежденческой среды и автоматизированных систем учрежденческой деятельности (АСУД).Посредством нее создается модель конкретного учреждения, которое следует автоматизировать. На базе этой модели проектируется управляющая среда АСУД.

MTA SZTAKI Tanulmányok 183/1986 pp 103-109

REQUIREMENT SPECIFICATION TECHNIQUES IN HARDWARE DESIGN

L.L. MÁTÉ

Computer and Automation Institute, HAS, Budapest
Hungary

Abstract

Some old and new examples of requirement oriented specification are shown first. A thorough analysis of the examples results that the hardware designer works on the basis of a set of elements available for him/her to satisfy the task.

The conceptual distance between the hardware to be designed and the set of useable elements is defined and estimated next. Due to very long and historically evolved and stabilized system of standardization this distance in hardware design is much shorter than that is in software development. Hardware standardizsation systems incorporate almost everything what has ever been designed and makes it available to the further designers as elements to be used and improved.

Finally refinement techniques applied on requirement specifications are analyzed and the role of nonformalized knowledge which is used during this development process is emphasized.

Concluding remarks close the paper.

1. INTRODUCTORY EXAMPLES

1.1. Ark[1]

"God saw that the wickedness of man was great in the earth, and it grieved his heart. He said upon Noah, 'Behold, I do bring a flood of waters upon the earth to destroy everithing that is in the earth. But, make thee an ark of gopher wood: the length of the ark shall be three hundred cubits, the breadth fifty cubits, and the heigth thirty cubits. Rooms shalt thou make in the ark, with lower, second and third stories. Pitch within and without with pitch, and set the door of the ark in the side thereof.'
And Noah did according unto all that Lord commended him."

14 Czinály magadnac Bárkát Gopher fából, ſoc haylokokat tſinály az Bárkában, és fenyö viaſſal meg öſſed azt belöl és kiuül.

15 Illyen g formára czinállyad pedig azt: Az Bárkának hoſſá három ſiáz ſing légyen, az ſiéleſſége légyen ötuen ſing, és az magaſſága harmintz ſing.

16 Világos ablakokat czinály à Bárkán, h és egy ſingnec nagyſága ſerint végezd el azt fellyöl, aytót pedig óldalúl czinály az Bárkán, álsó, közép, és harmad padláſoc légyenec benne.

17 Es imé én hozoc ez földre özön vizet, hogy elveſſeſſec minden teſtet, mellyben vagyon élö lélec az Eeg alatt, valami ez földön vagyon meg hal.

g: Az Noe Bárkájának formája.

h: Az az, Vgy czinálj az Bárkát, hogi egy ſing re emeld fel az Bárkának ſedelét: á hat mintz ſing feléött.

Figure 1. Application oriented requirement specification of Noah's ark as it appeared in the Hungarian translation of the Holy Bible translated and edited by Gáspár Károlyi, 1590, Vizsoly, Hungary

1.2. Calculating Engine[2]

"A proposition to reduce aritmetic to the domain of mechanism, - to substitute an automaton for a compositor, - to throw the powers of thought into wheelwork could not fail to awaken the attention of the world."

"Unlike all other machinery, the calculating mechanism produces, not the object of consumption, but the machinery by which that object may be made."

"For the sake of clearness, and to render ourselves more easily intelliglibe to the general reader, we have in the preceeding explanation thrown the mechanism into an arrangement somewhat different from that which is really adopted..."

"Mr. Babbage found, that, without some effective expedient by which he could at a glance see what every moving piece in the machinery was doing at each instant of time, such inconsistencies and obstructions as are here alluded to must continually have occured. This difficulty was removed by another invention of even more general nature than the calculating machinery itself, and pregnant with results probably of higher importance. This invention consisted in the contrivance of a scheme of mechanical notation which is generally applicable to all machinery whatsoever..."

BABBAGE' SZÁMOLÓ MOZGONYA.

Nincs a' társaságban irigyleudőbb helyezet, mint azon keveseké, kiknek mérseklett függetlensége felsőbb elmebeli tulajdonokkal van egybeköttetve. Kik mentek lévén azon kénytelenségtől hogy táplálatukat bizonyos életmód-választás által keressék, nincsenek annak nyügeivel korlátolva, képesek elmejök' erejét oda intézni, 's kirekesztőleg azon tárgyak körül egyesíteni, mellyekkel érzik hogy a' közhasznot leghathatósban előmozdíthatják 's magoknak legtartósb köz becsültetést szerezhetnek. Más részről közép állásponjok és határozott jövedelmök biztosítja őket a' hiúságnak 's tékozlásnak csábitásaitol, mellyeknek a' nagy jólet 's felsőbb rang mindig kiteszi ön biztosait. Illy

Figure 2. Fragment of Mr J. Győry's reporting on Babbage's Calculating Engine in the scientific quarterly "Tudománytár", Vol. 4., 1834, Buda, Hungary

1.3. Pocket Calculator[3]

"Shortly after the HP 9100 calculator, was announced in March 1968, William Hewlett expressed a desire that the next design be one-tenth the volume, and one-tenth the cost. After a trip to Japan in the summer of 1970 and a look at some of the portable four-function calculators being developed he reiterated his previous wish for a pocket size scientific calculator."

"On top of the requirement for high density and high speed was the need for low power. The size and volume specification was set by Helwett's shirt pocket, this meant few and small battereies. Four hours was set as goal for calculator operating time."

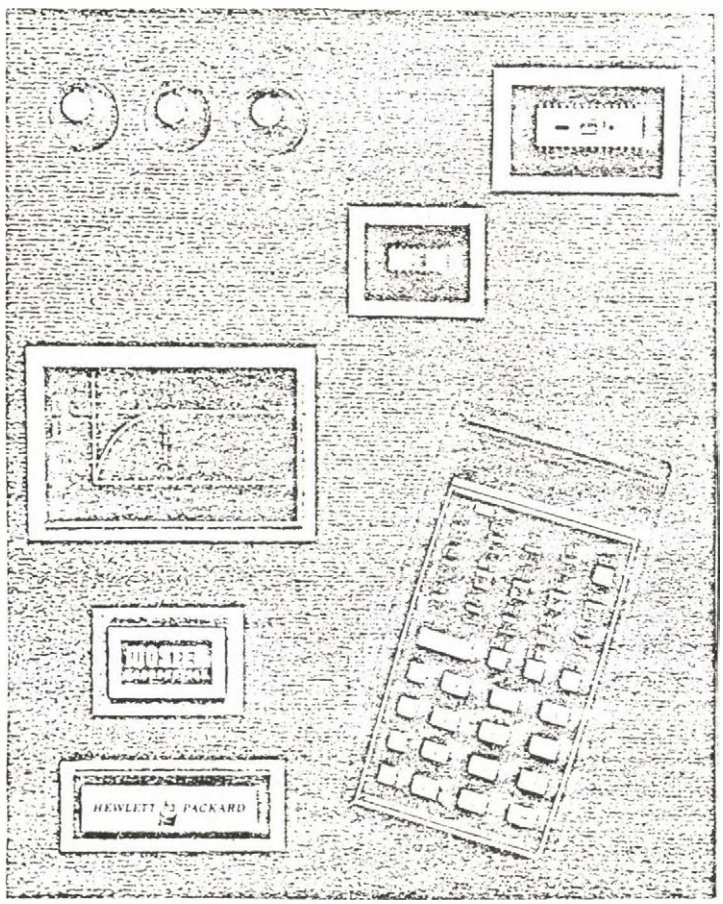


Figure 3. HP-35 as adverised in 1972

2. ANALYSIS

2.1. Ark

In the first example an Application Oriented Requirement Specification (AORS) is given in natural language.
When specifying the ark Lord answers the basic questions of AORSs - Why, What, How - in such a proper order that even Mr D. T. Ross[4] would be satisfied with that.
What is important here that both parties have some common knowledge, namely, both Lord and Noah know what the concept "ark" means. Describing Noah's task Lord refers to this common - pre-defined - concept and provides parameter values for it.
Parameters are referred by names (e.g. length, breadth, height, number of stories, place of door) values are given in standard units (e.g. cubit). Some implicit parameters - such as materials (e.g. gopher wood, pich) or even a certain "pitching" technology - are also referred. All parameters belong to the "knowledge base" common to both parties.
The number and comlexity of parameters completely defining the ark is very low. This fact can be interpreted as a very small conceptual distance between the primary concept of the AORS and its constituting elements due to high level structuredness of the knowledge base and an effective standardization the elements of the knowledge base are subjects of.

2.2. Calculating Engine

Our second example can not be interpreted as an AORS, nevertheless, it is included here to give us opportunity to illustrate some very general techniques of hardware AORSs. In the paper referred here the author describes how Babbage's Calculating Engine works, how it calculates table entries, how the typesetting mechanism produces masters for printing etc. These details are explained in English, but, this language is full of engineering and mechanical terms (e.g. wheel, axis, bolt, dial, index, wedge, etc.) which are very hard to understand for the unfamiliar reader.
The engineer has his own language for AORS. And that language is still not enough. As it can be seen in the third excerpt of 1.2. the author has described a model of the Calculating Engine instead of the real design. And this technique is also common practice among engineers. Models used during hardware design generally differ from the final practical design with respect to particular solutions, simplifications, redundancy, compactness etc. Elements of the model layer are not necessarily exsisting in reality, but those of the real layers are always real ones. This restriction does not mean that the later elements *must* be simple atomic ones, moreover, they can be subjects of further AORSs, as well. But, in a distinct period of the realization process even the composit elements will be treated as real elements and thus their inner characteristics can be forgotten and only

their behaviour - as their functioning can be seen from the outside - is taken into consideration.
An AORS for hardware design is well structured. The development of an AORS takes place in a top-down stepwise refinement manner from the most general ideae to the most detailed design. Number of layers used is arbitrary, but, due to high level of standardization it can be kept in most designs relatively low.
There are a few computer assisted systems to support the above approach. Here we refer our own system CARS[5] in the field of digital engineering.
The last paragraph in 1.2. deals with engineering notation. We are not intended to talk about the importance of notations for it is well known and practiced worldwide. But, it seems to be important to emphasize here that different fields of engineering have different notations. Usually, an engineering notation is well suited for the field it has been developed for, meanwhile, it is very unfit to other fields. Unifying efforts mostly have failed when artifical "common divisor" was to be developed for fairly distant fields of engineering. Notations used in AORS development are very field specific. We belive, differences between AORS development languages are results of natural evolution which should be understood, even aided, but, not fighted against.
One of the most promising approaches which can be succesfully adopted while aiding the evolution of field suited AORS development techniques is the Meta System approach[6] which gives common tools for different fields to ease the development of their own AORS languages.

2.3. Pocket Calculator

The great success of the HP-35 scientific calculator (the first of its kind in history) proves that a very clear highest level AORS increases the chances of achiveing the primary goals. However, top level AORS plays another role in the development process, as well. Its clearness, compactness helps to control the design by checking and evaluating partial results of development with regard to the original specification. Meanwhile, a lengthy and complicated AORS increases the danger of achieving some of the subgoals while missing the primary ones.
Some try to formalize the design process as a series of AORS transformations. However, it is quite obvious that formal tarnsformations of the top level specification given in 1.3. can never lead to the well known HP-35. Not even if the initial AORS were given in a more formalized way. Certainly, the developing process of any AORS into a refined, more detailed structure of AORSs can be treated as transformation. But the input for this transformation is not only the above level AORS, neither is it the formalized knowledge kept in the knowledge base alone, but a very important *non*formalized component - the design engineer's knowlege, experience and creativity - is also needed. Sorry to say, but, at least for a certain while, the design engineer can not be simplified out from the design process.

3. CONCLUDING REMARKS

Application Oriented Requirement Specification is a promising field of software engineering. Very long experience of hardware designers shows that standardization of concepts and solutions can have an effective influence on the development process. Well structured knowledge makes easier to use earlier results. Tools aiding the development of field specific notations seem to be also important. But, even formal transformations need human creativity and a great deal of nonformalized knowledge as major source of information in producing real results in AORS development.

4. REFERENCES

1 Moses, "Genesis" 6:13-16.

2 "Babbage's Calculating Machine", Edinburgh Review, Vol. LIX, No. CXX, 1834, pp. 263-327.

3 Whitney, Paluck, "MOS circuit development for the HP-35" COMPCON'72. pp. 73-76.

4 Ross, Schoman, "Structured analysis for requirement definition", IEEE Transaction on Software Engineering, SE-3, No. 1, January, 1977, pp. 6-15.

5. Máté et al., "System CARS and its description language" Lecture Notes on Computer Science, Vol. 152, 1983, pp. 1-16.

6. Demetrovics, Knuth, Radó, "Specification meta systems", Computers, Vol. 15, No. 5, May 1982, pp. 29-35.

MICROCOMPUTERS IN RETIREMENT SERVICE

L.L. MÁTÉ and *M. RUDA*

Computer and Automation Institute, HAS, Budapest
Hungary

Abstract

This paper is concerned with an interactive microcomputer system designed for assisting pension clerks in Hungary in the old-age pension determining process. Motivation comes from the huge size and complexity of the problem to be solved in an 8-bits microcomputer environment.

Hungarian Superannuation Act is a quite complex law-system. It determines pension as a sophisticated function dependent on

- the claimant's age,
- the cumulated length of the claimant's terms of service,
- the claimant's salary in a choosen five year period.

Moreover, some special allowances are given depending upon family status.
The pension function is periodically adjusted to inflation.

Since pension clerks are nonprofessionals with respect to computer utilisation our system obviously must be user friendly. The documents as produced should be very clearly drafted and logical for they are legal certificates. Common computer printouts (e.g. permitting capital English alphabet only) are not acceptable.

User friendliness is provided by our micro-SHIVA® - an easy-to-use form and data base management system.

The main tasks of the system are as follows:

- to aid the pension clerks in collecting certificates about the claimant's personal data, terms of office and salaries;
- to check consistency and completeness of the data collected;
- to run application programs of the pension function;
- to provide legal certificates and other documents for the claimant;
- to furnish statistical data;
- to produce archives of ready files.

The system is implemented on a Z80A based microcomputer of 64 kBytes with a CP/M compatible operating system.

1. MOTIVATION

The challenging offer came at the end of 1983: we were asked to develop microcomputer software for pension clerks.
An 8-bit microcomputer was given of 64 kBytes with Z80A microprocessor. A 2 x 700 kByte floppy and a matrix printer served as peripherial units. Our CP/M compatible Netty operating system and the almost finished micro-SHIVA™ form and data-base management systems[1] were given as tools and means of software development.

The callenge was manyfold:

- first application of the micro-SHIVA™ in nonprofessional environment,
- personal workstation for administrative personnel,
- legally proper official certificates by microcomputer,
- more than 100 installations expected in more than 30 places.

The offer was accepted.

2. PENSION FUNCTION

Hungarian Superannuation Act[2] is a quite comlex law system. It determines pension as a sophisticated function of

- the claimant's age,
- the cumulated length of the claimant's terms of office,
- the claimant's salary in a choosen five year period.

2.1. Age

Men are eligible for old-age pension at age 60, while, women at age 55 if having at least ten years of service. However, certain advantages are granted to persons working under hard conditions.

2.2. Terms in service

A co-efficient is defined as a function depending upon the claimant's cumulated terms of service. This co-efficient is called pension-rate and is determined as follows:
Let T be the cumulated terms of service in years. Then

$$\text{Pension-rate} = \begin{cases} 0 & \text{if } T < 10 \\ 13 + 2 \times T & \text{if } 10 < T < 25 \\ 38 + T & \text{if } 25 < T < 32 \\ 54 + 0.5 \times T & \text{if } 32 < T < 42 \\ 75 & \text{if } 42 < T \end{cases}$$

Terms of services should be legally certified. Individual terms of services are summed up with an accuracy of one day. Nonwork periods are omitted, but, longer than five year breaks restart summing. Age limit advantages are also counted here. Eligibility for old-age pension is decided upon the results of this summing up procedure.

2.3. Salary

The claimant can choose the best period from at most three five year terms to be evaluated:

- the last five years before retirement,

- the last five years before age limit,

- the last five years before age limit minus five years.

Salaries and premiums from the chosen period are evaluated. The best three years of the term is selected with respect to workday pay. Monthly average is calculated next from the best workday pay and this monthly average serves as the base-salary for the pension. Extreme high base-salaries are progressively reduced.

2.4. Pension

Base-pension is computed as the product of the base-salary and the pension-rate (the latter in %). Base-pensions under a yearly adjusted minimum amount are increased according to a set of rules. Premiums are given to late retiring blue collar workers. Special allowances are provided in the right of spouses in ceratain cases, and, family allowances are also given for dependants of the claimant.

Yearly compensation of inflation is also computed.

The above interpretation of the Hungarian Superannuation Act is very "computer minded" and oversimplified and is given here for just to render palpable the nonalgorithmic character of the law.

3. WORKSTATION

The pension clerk uses our system as her (it is a she almost without exception) personal workstation. She controls the whole process of determining a claimant's pension. The main tasks of this process are as follows:

- collect all the needed certificates about the claimant's personal data, terms of service and salaries;
- check consistency and completeness of data collected;
- compute pension-rate;
- compute base-salary;
- compute pension;
- produce legal certificates about the results for the claimant;
- furnish statistical data;
- update archieves.

For the above tasks the worstation provides her a data-base and a set of application programs. Everything is organized around her in a user friendly way. Since pension clerks are nonprofessionals with respect to computer utilisation, no computer expertness is needed in the usage of the system. Data is input via filling and/or modifying appropriate forms presented on the screen by micro-SHIVA® using standard typewriter keyboard. Special keys provide form manipulating functions e.g. turning over leaves of form, cursor moving among fields of form, queries by keys in the data-base, change of forms, etc. Even application programs are run by single keystrokes.

Menu and error messages are given on the screen in plain Hungarian and suggestions for nontrivial error corrections are shown on the screen as well.

Legal certificates are produced as modified hard copies of screen forms. Modification is needed for security purposes. By simple modifications the system assures that only semantically consistent documents can be printed with a "Valid" sign on them.

The documents are printed on the mosaic printer in plain Hungarian, using both upper and lower case characters, including all the special characters of the Hungarian alphabet, as well.

3. IMPLEMENTATION

3.1. Organisation

Data-base management, form compilation and manipulation mechanisms are provided by micro-SHIVA®. Application programs are written in PASCAL and interfaces between application programs and micro-SHIVA® are furnished by an interface package written in Z80 assembly. The general structure of an application program is the following:

- A top level PASCAL program supplies the set of names of forms useable by the given program. This restriction applies only to such programs which produce legal certificates.

- The top level Pascal program calls micro-SHIVA®.

- micro-SHIVA® works as a form and data-base manipulating tool while data input and/or modification takes place. It contols field types (e.g. alphabetic, integer, fixed point real, date etc.), thus syntax errors are shown at once.

- When striking the RUN key, micro-SHIVA® calls the appropriate application program which besides semantic control computes all results needed to fill the given form. When ready, it gives back control to Micro-SHIVA®, which shows the completed form.

- If the pension clerk is satisfied with the results she can put them into the data-base by a single keystroke. She can produce a hard copy of the resulting document on the printer, too. Certificate producing application programs do not allow modification on completed forms: should a single character be changed, the "Valid" sign is changed to "Void" automatically and this sign is unremoveable except at rerun of the application program.

- If work with the given claimant is finished, the pension clerk can continue either with another claimant with the same program, or with another program with the same claimant. Simultaneous change of program and claimant is also allowed.

The following application programs compose the software support of the workstation:

- Cumulated terms of service;
- Base-salary;
- Pension;
- Statistics;
- Correspondence;
- Archive.

3.2. Parameters

User friendliness also means relatively fast respond-time. Our system is quite satisfactory in this respect:

- in a query a negative answer is instantaneous, a positive answer including complete form filling takes about ten seconds;
- application program execution time is at maximum thirty seconds.

All application programs are implemented without overlay with the biggest needing 32 kBytes, thus, allowing sufficiently large form and data-base record sizes.

A one-floppy data-base can keep more than one hundred records, while a pension clerk handles about thirty cases per month.

4. CONCLUSIONS

Our system is already in intensive utilisation in two places in the country[3]. Experience shows that it satisfies previous expectations. However, the security provided by it is too tight and makes it a bit slower than expected. Documents are clear and well accepted. Althogether, it eases the work of pension clerks, makes the whole old-age pension determination faster and more accurate.

Further investigations are directed toward the utilisation of local area network facilities and data telecommunication means.

5. REFERENCES

1 Kerékfy, P., Ruda, M., "Micro-SHIVA user friendly information system", Lecture Notes in Medical Informatics, Vol. 24, Springer, 1984, p. 240.

2 "Hungarian Superannuation Act", (Act II. of the year 1975), Special edition for pension clerks in Hungarian, Budapest, 1983.

3 Máté L., Ruda M., "Determining old-age pension by micro-computer", PROGRAMOZASI RENDSZEREK'84, Szeged, 1984, p. 53.

DATA BASES

БАЗЫ ДАННЫХ

MTA SZTAKI Tanulmányok 183/1986 pp 119-125

A PROJECT FOR DATA INTEGRITY SUPPORT IN THE SYSTEM ML-1

P. STRANJEV, G. ANGELOVA

Laboratory of Mathematics Linguistics
Institute of Mathematics of BAN
Sofia, Bulgaria

SUMMARY

The paper describes a realization of a project for embeding some intelligence into the information-retrieval system ML-1, elaborated at the Laboratory of Mathematical Linguistics. The main purpose of the "intelligent" subsystem of ML-1 is to model the ML-1 actions whithout the end users' intervention.

A BRIEF DESCRIPTION OF THE SYSTEM ML-1

The system ML-1 represents a relational information-retrieval system enabling preparation of correspondence letters and documents by preliminarily given standart texts of the letters and documents. The system ML-1 is realized at the Laboratory of Mathematical Linguistics of the Institute of Mathematics for the computer SM-4 under the operating system DOS RV-B (compatible with PDP-11/40 under the operating system RSX11M) using the programming languages FORTRAN and MACRO-11. The system ML-1 can be generated over different data bases; the users' data are organized as relations, which are physically stored as indexed or sequential files. The users' requests are processed by algorithms for decomposition similar to the algorithms used in the system INGRES [1]. The system ML-1 provides a relational query language, similar to the query language QUEL [1]. The query language of the system ML-1 includes the standart query language operators FIND, ADD, MODIFY and DELETE. An addi-

tional operator for preparation of correspondence letters or documents is implemented. The operator has the following format:

PRINT A LETTER <number_of_a_letter>

FOR THE RELATION <name_of_a_relation>

WHERE <qualification> .

The standart text of the letters and documents is preliminarily entered into the current directory as a file named LET<number_of_the_letter>.DAT and it contains all data base elements which have to be extracted. The elements are described as sequences from the type @<name_of_a_rel>.<name_of_an_attribute_of_the_rel>.

The operator PRINT A LETTER lists the elements mentioned in the standart letter text, extracts the necessary data from the data base and forms the corresponding number concrete letters with their addresses. Each letter is formated and edited in accordance with the bulgarian grammatical rules. So the user has the possibility to form an unlimited number letters' and documents' types.

Fig.1 presents a functional scheme of the system ML-1.

ABOUT THE NEEDS OF AUTOMATIZATION OF THE END USERS' ACTIVITIES

The end user often performs some activities related to data base support (for instance, the adding and modification of the data). During the process of introducing new values into the data base some control over these values is needed. It is possible to check the new data using the so called "integrity constraints" describing data values dependences [2].

The end user of the system ML-1 often prepares letters

and documents when preliminarily given conditions concerning data base values arise. Many letters' types and the corresponding conditions can be specified at the stage of data base design. Many aspects of the activities mentioned above can be formalized and suitable algorithmisized in order to facilitate the end user interaction with an information-retrieval system.

THE "INTELLIGENT" SUBSYSTEM OF ML-1 AND ITS IMPLEMENTATION

The "intelligent" subsystem of ML-1 is activated when the user adds new data or modifies the data values from the data base. Thus the main purpose of the project for an "intelligent" subsystem realization is to assure the checking of some preliminarily given conditions concerning data base values, to generate appropriate operators from the ML-1 query language and to control the execution of these automatically stored operators.

Fig.2 presents a functional scheme of the system ML-1 and its "intelligent" subsystem.

The "intelligent" subsystem of ML-1 uses a knowledge base containing production rules [3] from the type:

IF <expression> THEN <description of an action>
ELSE <description of an action>.

The expression represents an arithmetical or logical expression concerning data values from the data base and data values from the logical data base scheme.

The action can represent one of the following operators:

-GENERATE THE OPERATOR ADD...
MODIFY...
PRINT A LETTER...

-CONTINUE ;
-PRINT <a text of a message> ;
-RETURN .

The operators ADD,MODIFY and PRINT A LETTER are stored into an intermediate buffer in accordance with the ML-1 query language syntax.

For example, if the user tries to add and modify the field SUPPLIER.SALARY, the following rule can be applied:

```
IF (SUPPLIER.SALARY > 120 ) AND (SUPPLIER.SALARY < 500)
                        THEN CONTINUE
                        ELSE PRINT 'THE VALUE IS WRONG';
                        RETURN.
```

Thus the so called "integrity constraints" can be embeded as production rules into the knowledge base.

The additional operators ADD and MODIFY enables some actions supporting the data integrity; the operator PRINT A LETTER assures the signal correspondence preparation. The operators from the intermediate buffer can be activated by a special users' request; the control over the execution is performed by the submonitor of the additional requests (see fig. 2).

The submonitor servicing the knowledge base enables to the data administrator to list, add, modify and delete the rules from the knowledge base.

CONCLUSION

The "intelligent" subsystem of ML-1 can ensure some aspects of data integrity whithout users' interventions and automatize the documentary activities.

The described more intelligent version of the system ML-1 is oriented towards users - nonprogrammers and towards application areas having not very large data bases.

REFERENCES

1. Stonebraker M., Wong E., Kreps P., Held G. The design and implementation of INGRES. ACM Trans. on Database Systems, 1:3, 1976, p. 189 - 222 .
2. Ullman J. Principles of Database System. Stanford University, Computer Science Press, 1982 .
3. Nilsson N.J. Principles of Artificial Intelligence. Springer - Verlag, 1982.

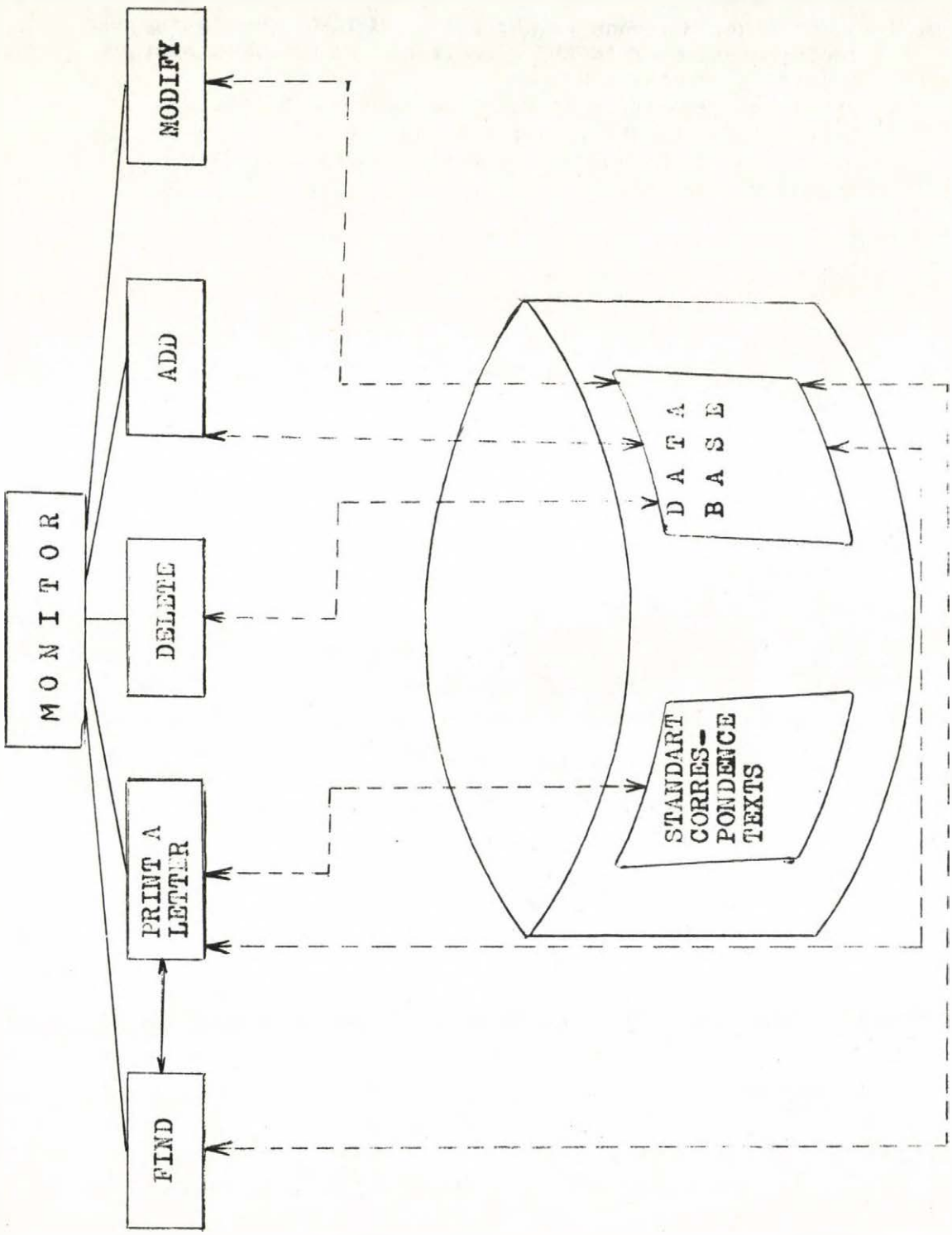


Fig. 1. A functional scheme of the system ML-1

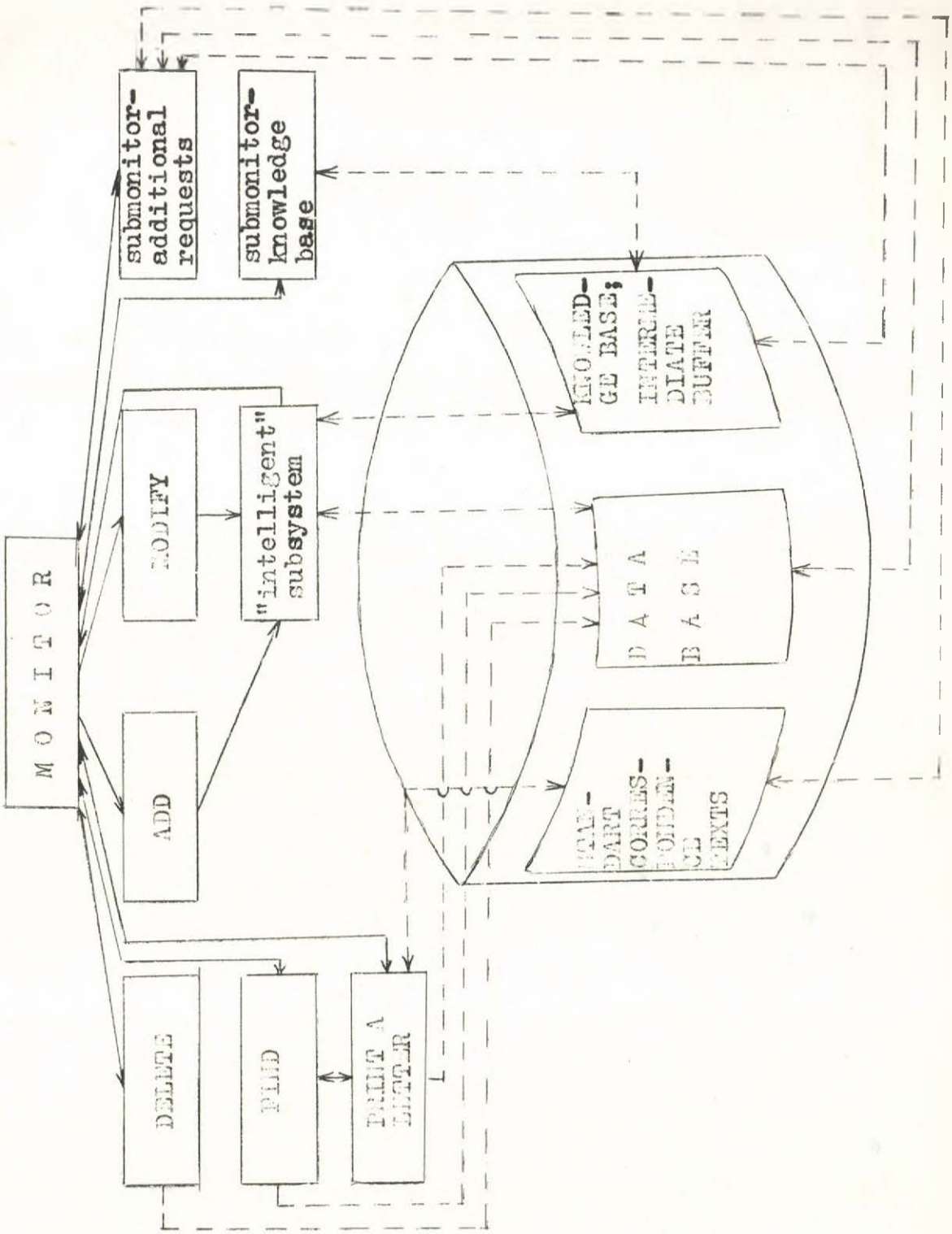


Fig. 2. A functional scheme of the system ML-1 and its "intelligent" subsystem.

MTA SZTAKI Tanulmányok 183/1986 pp 127-134

СРЕДСТВА ИНФОРМАЦИОННОГО ПОИСКА И МАНИПУЛИРОВАНИЯ ДАННЫМИ В ОС BOS2

Румяна Лесева
Единный центр математики и механики,
Болгарская Академия наук

Рассмотрим стандартные средства, предосталяемые операционной системой BOS2 для обеспечения информационного поиска, манипулирования данными и генерирования отчетов.

Некоторые программы обрабатывают файлы произвольной структуры, а другие требуют определенного структурирования обрабатываемых файлов. Так что дальше будем рассматривать применение программ для файлов следующей структуры:

Файл состоит из записей, оканчиваемых разделителем записей. Максимальная длина записи 512 байтов. Записи состоят из полей, ограниченных разделителями полей. Число полей не может быть больше 50.

Физическая структура данных соответствует логической структуре данных в реляционной базе данных, если поставим в соответствии реляции - файл, кортежу - запись, атрибуту - поле. Так что ниже будем говорить о файле, как и о таблице.

Вся информация в символьном формате, что не мешает выполнять над ней операции как символьного, так и арифметического типа в зависимости от контекста.

Основное средство операционной системы для создания фай-

лов - текстовый редактор ed . Он дает возможность включения, удаления, изменения строк, как и подстановки текстов, задаваемых регулярными выражениями.

Над файлами (таблицами) вышеупомянутой структуры можно выполнять следующие операции:

- селекция записей, удовлетворяющих заданными условиями
- соединение таблиц по совпадению значений указанных полей
- проекция таблицы, т.е. таблица, содержащая подмножество столбцев исходной таблицы
- объединение, сечение, разница таблиц, рассматриваемых как множества строк
- обновление таблиц, т.е. включение, удаление и модификация записи или группы записей.

Над записями (строками таблиц) можно выполнять операции:

- включение записи (записей)
- обновление записей
- удаление записи

Поля записи (столбца таблицы) не могут быть больше 50. На них можно ссылаться, они идентифицируются своим порядковым номером внутри записи - § 1 по § 50.

Над полями можно выполнять операции сравнения (==,!=,>=, >,<=,<) с выражениями, содержащими константы, переменные и значения полей; операции соответствия, задаваемого при помощи регулярных выражений; арифметические, логические и строчные операции; применять некоторые стандартные функции.

Рассмотрим более подробно основные функциональные возможности, предоставляемые средствами ОС BOS 2 для информационного поиска, манипулирования данными и генерации отчетов.

Приведем примеры на основе трех таблиц, содержащих инфор-

мацию о поставщиках и деталях. Содержание таблиц и их полей следующее:

таблица S - информация о поставщиках
- 1 - номер поставщика
- 2 - имя поставщика
- 3 - статус поставщика
- 4 - город поставщика

таблица P - информация о поставляемых деталях
- 1 - номер детали
- 2 - наименование детали
- 3 - цвет детали
- 4 - вес детали
- 5 - место производства

таблица SP- информация о поставщиках и поставляемых ими деталях
- 1 - номер поставщика
- 2 - номер детали
- 3 - поставляемое количество

Ниже дано примерное содержание таблиц:

```
S:  s1 Smith 20 London
    s2 Jones 10 Paris
    s3 Blake 30 Paris
    s4 Clark 20 London
    s5 Adams 30 Athens

P:  p1 nut red 12 London
    p2 bolt green 17 Paris
    p3 screw blue 17 Rome
    p4 screw red 14 London
    p5 cam blue 12 Paris
    p6 cog red 19 London
```

```
SP: s1 p1 300
    s2 p2 200
    s1 p3 400
    s1 p4 200
    s1 p5 100
    s1 p6 100
    s2 p1 300
    s2 p2 400
    s3 p2 200
    s4 p2 200
    s4 p4 300
    s4 p4 300
    s4 p5 400
```

Селекция записей выполняется программой **awk**.

При задании условий селекции (шаблонов) используются операторы сравнения (==, !=, >=, > , <=, <); соответствия (~, !~), которые задаются регулярными выражениями и логические операторы (&&, || , !). Как операнды могут участвовать константы, переменные и поля. Селекция осуществляется путем проверки для каждой записи удовлетворяет ли она заданными условиями.

Пример: Вывести информацию о всех поставщиках, чей город Paris и статус больше 20. Это выполняется **awk** программой над таблицей **S** :

```
awk '$4~/Paris/&& $3>20 {print} ' S
```

Результат следующий:

```
s3 Blake 30 Paris
```

Проекция осуществляется путем включения в выходной файл только указанных полей выбранных записей.

Пример: Найти имена и статус всех поставщиков из города London . Это можно получить **awk** программой, обрабатывающей таблицу **S**

```
awk '$4=="London" {print $2 $3} ' S
```

Результат выглядит так:

```
Smith 20
Clark 20
```

Результат проекции может содержать повторяющиеся записи. Если необходимо удалить их, то это можно сделать при помощи команды **uniq** , которая в сортированном файле устраняет повторяющиеся записи.

Квантификация задается неявно, так как селекция выполняется путем проверки для каждой записи выполняет ли она заданные условия. Таким способом можно подсчитать разные количества, как например общее число, сумму, среднее количество и др.

Пример: Получить список номеров всех деталей, поставляемых более чем одним поставщиком, что выполняется awk программой при использовании ассоциативных массивов. Информация извлекается из таблицы SP .

```
awk '{p[§2]++}
     END {for ( i in p )
            if ( p[i]> 1 )
              print i} ' SP
```

Результат выглядит так:

```
p1
p2
p4
p5
```

Соединение таблиц осуществляется командой join. Две таблицы могут быть соединены по совпадению содержимого указанных полей. Таблицы должны быть сортированы относительно соответствующих полей. Команда join разрешает вместе со соединением выполнять и проекцию.

Пример: Получить имена всех поставщиков, не поставляющих деталь p1.

```
join -o 1.2 2.2 SP S > SSP1
awk '§1!~/p1/  {print §2} ' SSP1 > SSP2
uniq SSP2
```

Здесь команда join выполняет соединение таблиц SP и S по первому полю (номер поставщика). Результатные записи содержат только второе поле записи SP (номер детали) и второе поле записи S (имя поставщика), так что результатная таблица SSP1 является проекцией соединения. Команда awk выбирает записи, у которых первое поле отлично от p1 и помещает их в таблицу SSP2 . Команда uniq

удаляет повторяющиеся записи. Результат действия перечисленных команд следующий:

```
Blake
Clark
```

Сортирование выполняется командой **sort**. Сортирование осуществляется по содержимому полей (от §1 до §12) во возрастающем или убывающем порядке.

Над таблицами, совместимыми относительно набора столбцев (полей), можно выполнять множественные операции:

- объединение произвольного числа таблиц в заданном порядке выполняется командой **cat**
- пересечение и разница двух таблиц осуществляется при помощи команды **comm**.

Над выбранными при селекции данными можно выполнять арифметические и строчные операции программой **awk**.

Пример: Для каждой детали найти вес общего поставляемого количества:

```
awk ' { p[§2]+=§3}
      END  {for ( i in p )
                print i p[i]} ' SP > SSP1
join -a1.1 1.2 2.4 SSP1 P > SSP2
awk ' {print §1 §2 §3} 'SSP2 > SSP
```

Первая команда **awk** вырабатывает таблицу **SSP1**, каждая ее запись содержит номер детали и общее поставляемое количество. Команда **join** соединяет таблицы **SSP1** и **P**, включая в результатную таблицу номер детали, общее количество и вес единичной детали. Вторая команда **awk** перемножает общее количество на вес детали и включает в результатную таблицу номер детали и общий вес.

Результат работы вышеуказанных команд следующий:

```
p1   7200
p2  17000
p3   6800
p4  10800
p5   6000
p6   1900
```

Можно выполнять группирование записей по признаку, общему для группы (содержимое поля или полей, которое удовлетворяет определенным условием). Записи групп можно помещать в новые таблицы для последующей обработки или подсчитать над ними разные функции: число, среднее число, максимум, минимум, сумму и др.

Пример: Найти число поставщиков для каждой детали. Это реализируется следующей awk программой:

```
awk 'BEGIN  {print "p§  count"}
            {s[§2]++}
     END    {for ( i in s )
                printf "%s  %s\n",i,s[i]} ' SP
```

Результат выполнения программы следующий:

```
p§  count
p1  2
p2  4
p3  1
p4  2
p5  2
p6  1
```

Программа awk предоставляет большой набор средств для редактирования и форматирования содержимое таблиц, что дает возможность на основе выбранной и перерабатанной информации генерировать отчеты.

Обновление хранимых в таблицах данных осуществляется в основном при помощи редактора текстов ed и программы awk. Можно выполнять включение, удаление и модификацию записей, выполняющих за данные условия.

Перечисленные стандартные средства операционной системы BOS2 предоставляет пользователю большие и гибкие возможности как для облегчения его повседневной работы, так и для решения более крупных задач информационного поиска, манипулирования данными и генерации отчетов.

ЛИТЕРАТУРА:

1. Документация операционной системы BOS2, ЕЦММ, БАН, 1984
2. Date C. J. An Introduction to Database Systems, Addison Wesley Publishing Co., Reading, MA, 1977

INFORMATION RETRIEVAL AND DATA MANIPULATION IN BOS2 OPERATING SYSTEM

R. Lesseva

Abstract

The tools of the BOS2 operating system for information retrieval, data manipulation and report generation are presented The required file structure is outlined. Implementation of some basic operations of information servicing using these tools is concerned and some examples are supplied.

MTA SZTAKI Tanulmányok 183/1986 pp 135-141

ОРГАНИЗАЦИЯ И ОБРАБОТКА СОЦИАЛЬНОЙ ИНФОРМАЦИИ ДЛЯ БОЛГАРСКИХ УЧРЕЖДЕНИЙ

А.ПЕТКОВ, А.ТЕРЗИЕВ
Институт математики с ВЦ
Болгарская Академия Наук

При создании систем для обработки социальной информации чаще всего используется один основной ключ, который связан с данными социального индивида (человека). Для нужд машинной обработки был введен единый ключ для всех граждан Болгарии, названный единным гражданским номером (е.г.н.). Каждому гражданину соответствует уникальный номер, содержащий десять десятичных цифр и имеющий следующую структуру:

ггммддхххх

Первая группа этой структуры состоит из двух цифр, представляющих год рождения индивида. Вторая группа указывает месяц. Третья группа указывает день рожденья, а четвертая представляет собой порядковый номер.

При создании информационных систем для социальных нужд в качестве основного ключа опроса и актуализации информации используется е.г.н. В таких системах широко используется два способа организации данных: индексно-последовательная и директная организация с рандомизацией при помощи выбранной хеш-функции. В обоих случаях каждому индивиду соответствует одна запись в информационном файле. Анализ распределения в этих случаях показывает что оно неравномерное т.к. основную роль в формировании е.г.н. исполняет дата рождения. При большинстве систем с конкретной социальной направленостью существует интервал активности, при котором число

регистрированных в систем индивидов значительно больше. Кроме того, интервал активности передвигается с временем т.к. для одних индивидов наступает период данной социальной активности, а для других индивидов этот период кончается. Возмем как пример структуру информации, связанной с системой для регистрации и отчета водителей транспортных средств.

В этих случаях теоритической границей возраста является период с 15-ти лет до максимальной продольжительности человеческой жизни, т.е. до 100 лет. На практике однако, большинство правоспособных водителей находится в возрастной границе с 18-ти лет до 66-ти лет. При этом, вне этих границ наблюдается небольшое число исключений.

При неудачном проектировании системы с стандартной организацией существует опасность неравномерного распределения данных в файле, появления данных в файле, переполнения области или удолжения времени обработок. При проектировании системы для обработки социальной информации необходимо иметь ввиду конкретную направленность системи и в соответствие с нею распределение регистрированных индивидов по группам, от которого зависит и физическое распределение записей данных в файле.

В настоящей работе предлагается способ организации данных для информационных систем с узкой социальной направленостью, обеспечивающий создание насыщенного интервала активности, который изменяется со временем. Во всех случаях е.г.н. является ключом (идентификатором) записей. Предлагаемая организация является индексной, но создается на файлах, для которых возможен директный доступ.

В системе поддерживаются два уровня индексов: главный и вторичный индекс (фиг.1).

Главный индекс содержит указатели к вторичным индексам и структурирован в группах по 12 указателей. Каждый указатель, принадлежащий к данной группе соответствует одному месяцу года. Число групп в главном индексе ровно сумме величины активного интервала при конкретной социальной активности и величины начальной нулевой группы, содержащей указатели к исключениям, которые в текущем году находятся вне активного интервала.

Кажды вторичный индекс представляет собой группу из 31 указателя к цепочке. Цепочка представляет собой однонаправленную последовательность записей с данными для всех индивидов, регистрированных в системе и рожденных в один и тот же день. Каждая запись содержит указатель к следующей записи в ее цепочке. Последная запись содержит признак конца цепочки. Т.к. число людей, рожденных в один и тот же день считается статистически одинаковым, то можно ожидать, что все цепочки будут с приблизительно равными длинами.

Поиск определенной записи по данному ключу, которой представляет собой единый гражданский номер, состоит из следующих этапов:

- вычисление определенной записи в главном индексе, который непрерывно находится в оперативной памяти. Группа элемента (год в интервале) вычисляется по формуле:

иг= (АГ-АД) - (Тг-ГГ-АД), где

иг - год в интевале

АГ - верхняя возрастная граница активного интервала;

АД - нижняя возрастная граница;

ГГ - год рождения в е.г.н. ;

тг - текущий год.

Если иг > АГ–АД или ИГ ≤ 0, то это означает, что индивид относится к группе исключений и полагается иг=0.

Номер элемента в главном индексе вычисляется следующим образом:

не= иг.12 + ММ- 1

При этом считается, что элементы номерованы от 0 до (АГ–АД+1).12 – 1

- чтение соответствующего вторичного индекса и от него – указателя к цепочке;
- последовательный просмотр цепочки от начала до конца для определения заданного ключа.

Быстродействие поиска в основном определяется числом входно-выходных операций с диском. Если допустим, что в одной системе регистрированы 100 000 индивидов для активного интервала, содержащего 50 лет, то это означает что средняя длина цепочек будет около пяти–шести записей. При подходящем оптимальном физическом расположении записей операция по поиску будет содержать только одно перемещение головок и в среднем около трех–четырех последовательных чтений записей. Если файл не реорганизован удачно и записи из одной цепочки физически разбросаны на разные места в файле, то перед каждым чтением будет совершаться соответствующее перемещение головок. Из-за этого в начале каждого календарного года необходимо реорганизовать файл. При этом записи из цепочек упорядочиваются физически последовательно.

Процесс реорганизации включает и перемещение активного интервала, что отражается в главном индексе следующим способом:

- группа, которая находится до начальной группы исключений, переходит к ней (т.е. самый старший год выходит вне интервала);

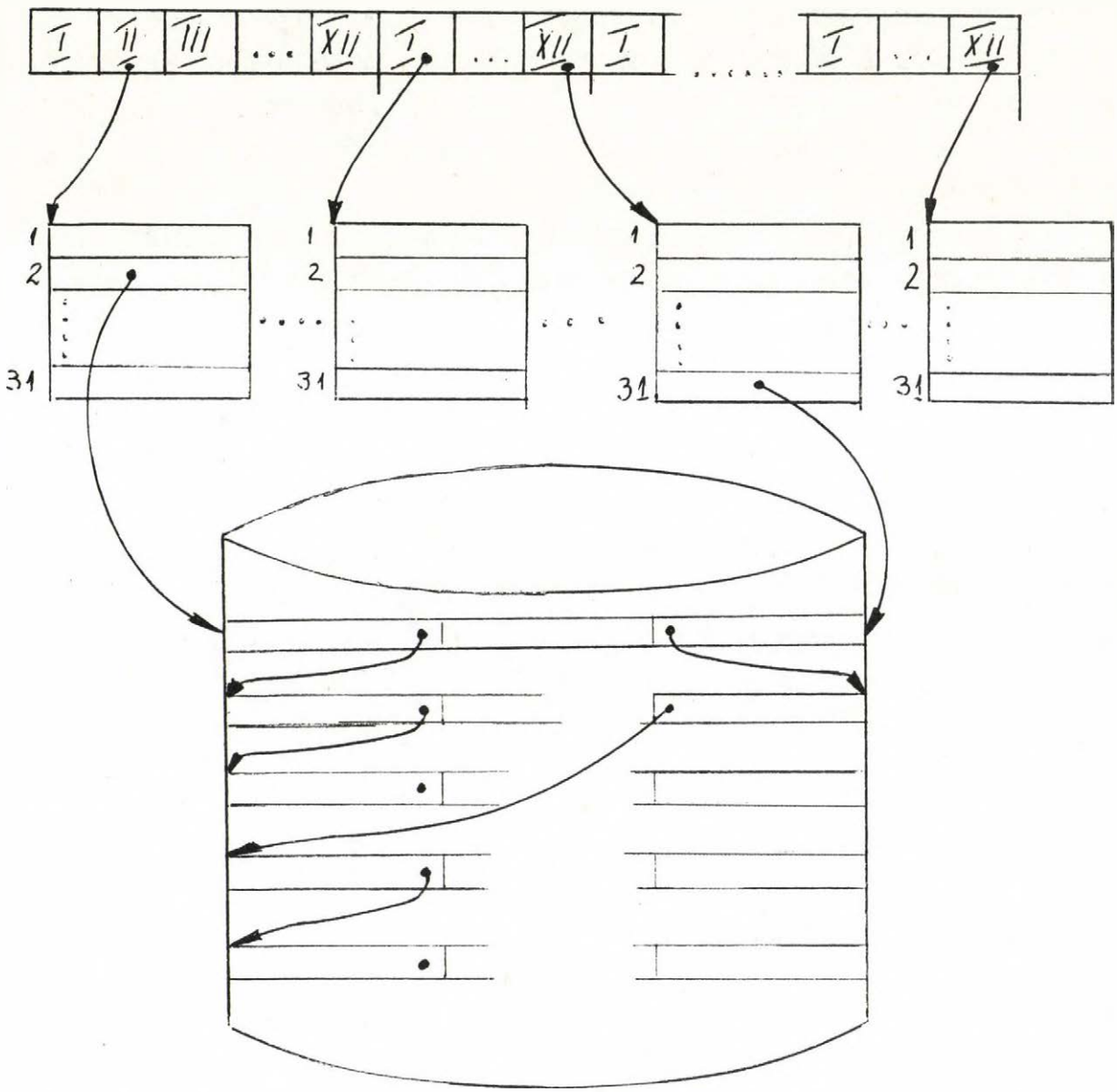
– все остальные группы перемещаются с одной налево,(т.е. интервал передвигается с одним годом вперед);

– справа появляется новая группа, отвечающая новому году, который входит в интервал, и к которой, перебрасывается соответствующие ей данные, составляющие до этого момента частью группы исключений.

Предлагаемая организация будет удобной и в том случае,когда система реализуется на запоминающих устройствах на магнитном диске с капацитетом носителей, недостаточным чтобы сохранить всю информацию. В этот случай элементы главного индекса могут содержать номер носителя, который содержит вторичный индекс и связанные с ним записи.

Описанная организация была реализована в системе регистрации и отчета водителей автотранспорта для нужд КАТ. Эта система предназначена для работы с миникомпютером СМ-4 в двух вариантах: с капацитетом дисков соответно 2,2 Мбайта и 21 Мбайта.

Эксплуатационный опыт показал что и на практике предлагаемая организация сокращает время доступа к данным.



фиг. 1

STRUCTURE AND PROCESSING OF SOCIAL INFORMATION IN BULGARIAN OFFICES

A.Petkov, A.Terziev

Institute of Mathematics with Computer Center, BAS

Abstract

A data structure and an access method for compressed data saving and retrieval of social information are presented. A comparison of different kind of data access and structure is made. An estimation of the proposed solution is given and its advantages are outlined. In particular, the data processing system for account of driving licences and other related data is described.

MTA SZTAKI Tanulmányok 183/1986 pp 143-148

ЗАЩИТА ДАННЫХ И АВТОРИЗАЦИИ ДОСТУПА В СИСТЕМЕ ОБРАБОТКИ ДАННЫХ

Атанас Терзиев
Институт Математики с ВЦ, Б А Н

В базах данных, рассчитанных на многие пользователи, должны быть предусмотрены средства, позволяющие пользователям совместно использовать данные, и в то же время обеспечивающие защиту данных от несанкционированного доступа. В методологии проектирования баз данных давно уже особое внимание удаляется защите /4,5/. Иерархия в представлении и модели данных привела к иерархии в полномочии и привилегий доступа /1,6/. Еще с появлением СУБД начали попытки применения авторизации доступа к коллективно использованным даннам /2/. Были разработаны и языки для описания защиты и безопасности данных /3/.

В этом докладе рассматривается механизм защиты данных в системе ДАКОМС /7/. Представлен язык для описания полномочий пользователей и защиты доступа.

Система ДАКОМС предоставляет средства интегрирования гетерогенных баз данных. В связи с этим, представление данных баз данных во время обработки (при помощи специализированных "коммуникационных" модулей) трансформируется во внутренное виртуальное табличное представление.

Для системы характерна многоуровневая защита данных. Все уровни опциональные, т.е. во время описания данных их владельцы могут потребовать той или другой защиты. На нижнем уровне находится физическая защита, т.е. данные хранятся во внешном накопителе таким образом, что они недоступны класическим средствам операционной системы. Это осуществляется на основе модификации физи-

ческой системы ввода/вывода по определенному закону и параметрам, указанным при описании концептуальной локальной модели (схем).

Следующий уровень – криптография. Во время определения данных можно вводить параметры для системного шифрования и дешифрования данных в процессе ввода/вывода. Есть возможность включить пользовательские программы, поддерживающие этот процесс по стандартизованному интерфейсу.

При решении проблемы доступа к элементам базы данных предусмотрен целый ряд факторов. Защита информации в зависимости от значений данных осуществляется на основе подсхем. Различные подсхемы связаны с разными пользователями. В таком случае у каждого пользователя будет собственное представление о базе данных и в нем будут отсутствовать сведения о недоступных для него данных.

Доступ, зависящий от полномочий реализован на уровне виртуальных таблиц. Администратор баз данных, владелец данных или уполномоченный пользователь предоставляет права доступа к элементу подсхемы и степени авторизованности к использованию значений данных. Для этой цели используется следующий оператор языка описания защиты:

```
DEFINE SECURITY=name PASSWORD=pasname DICTIONARY=dicname

        ident_i = { READ   } (mask_i)
                  { WRITE  }
                  { UPDATE }

        ident_2 = . . . . .

    END
```

dicname определяет словарь данных, по которому будут задаваться элементы подсхемной виртуальной таблицы. В ДАКОМС есть возможность работать одновременно с несколькими группами обозначений элемен-

тов данных.

$ident_i$ задает имя элемента, для которого определяется тип доступа Если некоторые элементы не заданы, то считается, что доступ к ним запрещен.

$mask_i$ представляет маска доступа, указывающая на уровень авторизованности доступа к описываемому элементу. При определении схемы данных некоторые элементы могут быть TYPE = SECURITY т.е. их содержимое представляет "замок" для доступа к другому элементу. Во время доступа к элементу и к экземпляру виртуальной таблицы модуль "управление доступа" проверяет "входит" ли в замок $mask_i$ и предоставляет стоимость этого элемента пользователю, или генерирует пустое значение (чтобы сохранить структуру виртуальной таблицы). Если $ident_i$ является ключом виртуальной записи, то тогда при неавторизованости по маске эта запись становится недоступной. Если экземпляр $ident_i$ получает значение во время доступа к нему, то тогда в соответствующем элементе типа SECURITY заносится и маска $mask_i$ авторизованности.

Важно отметить, что право собственности меняется в зависимости от условий управления баз данных. Для того, чтобы связать атрибуты защиты с конкретными пользователями, в системе ДАКОМС сохраняются профили пользователей. Они получаются вследствие трансляции оператора:

```
DEFINE  USER=userid STARTPROC=sproc ENDPROC=eproc
                  { USER          }
        TYPE  =   { OWNER         }  PASSWORD=... TRACE=....
                  { ADMINISTRATOR }

                  { ONLY } { PROC }
        USE       { NOT  } { MOD  }  (список имени)

        USE  .....
```

$$\left\{\begin{matrix}\text{CREATE}\\ \text{DELETE}\\ \text{UPDATE}\end{matrix}\right\} = \left\{\begin{matrix}\text{TEMPORARY}\\ \text{PERMANENT}\end{matrix}\right\} \left\{\begin{matrix}\text{USER}\\ \text{DICTIONARY}\\ \text{SCHEME}\\ \text{SUBSCEME}\\ \text{SOURCE}\\ \text{PROCEDURE}\\ \text{SECURITY}\\ \text{POST BOX}\\ \text{VARIABLE}\end{matrix}\right\} \left(\left\{\begin{matrix}\text{ALL}\\ \text{prefix}\end{matrix}\right\}, \text{param}\ldots\right)$$

. . . . . .

END

Монитор системы управляет все транзакции и активирует пользовательские программы. Он является и самым верхним уровнем защиты. Система открыта и к ней можно подключать разные проблемные пакеты программ. Все пользовательские программы предназначены для совместного использования многими пользователями. Часто необходимо запретить некоторым людям выполнять определенные обработки, программы или процедуры (каталогизированная последовательность программ и/или операторов). Подоператор USE... заносит в профиль пользователя список допустимых/недопустимых программ/процедур. В списке можно задавать общий префикс группы программ/процедур в виде: префиксжжж. При назначении полномочий можно указать, какие операции (CREATE, DELETE, UPDATE) разрешены с объектами системы (USER, DICTIONARY, ...) и/или их компонентами ($param_i$).

Монитор следит за обращением к программам и обработкам,и, если указан TRACE, то он ведет учет об их использовании, запрещает или выполняет их.

При подключении пользователя к системе монитор загружает в оперативной памяти его профиль. Если там указана STARTPROC, то

он автоматично загружает процедуру и выполняет ее, после чего пользователь может выполнять обработки. При окончании работы выполняется процедуру eproc (если она дефинированна). При дефинировании процедур можно указывать условия их выполнения.

Благодаря всему этому можно создавать хорошие средства обработки для конечных пользователей, используя коллективный доступ к даннам и средствам обработки, обеспечивая надежную защиту и авторизованность доступа.

ЛИТЕРАТУРА:

I. Bussolati U.,Martella G. "Access control and management in multilevel DB models", LNCS, v.I23, I98I

2. Griffiths P. "An Authorization Mechanism for a Relational DB System", ACM Transaction on DB Systems,v.I, N 3, I976

3. Hartson H.R. "Languages for Specifying Protection Requirements in DB Systems", Ph.D. dissertation,The Ohio State University

4. Hoffman L.J. "Modern Methods for Computer Security and Privacy", Prentice Hall I979

5. Hsiao D.K. "Computer Security", Academic Press; I979

6. Minsky N. "International Resolution of Privacy Protection in DB Systems", Communications of ACM, v.II9, N 3, I976

7. Терзиев А. "Интегрирование данных гетерогенных систем", MTA SZTAKI Tanulmanyok I47/I983, I983

DATA SECURITY AND AUTHORIZATION OF ACCESS IN A DATA PROCESSING SYSTEM

Atanas Terziev
Institute of Mathematics with Computer Center

Abstract

A formal approach is proposed for the definition and maintainance of data security and privacy in a data processing system. The levels of protection, the method and mechanism to provide the authorization of access and protection control are considered. In particular, a description of protection and data security in the DACOMS database system is presented.

MTA SZTAKI Tanulmányok 183/1986 pp 149-155

USER FRIENDLY DATA MANAGEMENT TOOLS FOR GENERAL APPLICATIONS ON MICROCOMPUTERS

P. KERÉKFY and *M. RUDA*

Computer and Automation Institute
Hungarian Academy of Sciences
Budapest, Hungary

Summary

The authors do not intend to present a fundamentally new conception in software development. The paper is connected with software production on user level. Possibilities offered by the utilization of microcomputers are discussed. One of our aims is to support development of data management systems on cheap microcomputers for unexperienced users with high level of data security. The visual man-machine connection is emphasized.

Fields: Software development on user level.

Keywords: microcomputer, user-level software development, visual man-machine connection, data security, full-screen editor under CP/M, automatic programming.

1. Introduction

This paper is a contribution to the matter of automatic programming, laying emphasis on miçrocomputer applications. The authors aspired to develop program generator procedures. Our conception differs from the very high level languages addressed to advanced programmers and from the knowledge-based systems specialised for a narrow field of tasks and utilizations. In the computer applications the possibility of different interpretations, modifications and different methods of applications is inherent in the intellectual work of end users. Consequently, tools are needed that support the software development on user level.

First of our results in this field are applied in large-scaled and complex statistical systems.

The most important features of this system /SIS79-GENERA/:

1. Fast and flexible data transmission and conversion.
2. Optimized /compressed/ data storage.
3. Data transformation and representation of network structures by graphs.
4. Evaluation of complex logical expressions.
5. Special optimization methods for data processing in descriptive statistics /selections, statistical tables, etc./.

Below we describe the realisation of the same aspirations on microcomputers.

2. A data management tool on microcomputers

Our purpose is the presentation of the fast, flexible and reliable use of microcomputers in data management with high level of data security. Results are based on development of several medical information systems.

The data management system developed for medical applications on small microcomputers is controlled by a monitor called Micro-SHIVA. The heart of this system is an extended full-screen editor that offers the following services:

- editing input/output forms and reports,
- definition of input/output fields in forms,
- data base I/O by means of forms, validity of data is checked automatically,
- definition, modification and query of data dase structure by means of forms,
- data query by special forms.

Now, let us see the role of the form editor subsystem in user-level system development. Our aims are the following:

Be unexperienced users able to utilize the system.

Man-machine connection be simple, easy to survey and fast.

Drawback caused by low capacity of cheap micros be compensated.

The activity of the unexperienced user can be best supported by a "visual man-machine connection". The connection is maintained by easy-to-survey forms /data forms, tables, figures, menus/ displayed on the screen.

"Full-screen" management of these forms facilitates the dynamic man-machine connection. It is the task of the form editing subsystem.

Form editor of Micro-SHIVA is a full-screen editor offering the following services:

1. Usual cursor moving, inserting, deleting, tabulating.
2. Structured text file. A file may consist of up to 255 logical pages, a page - 255 lines, and a line 250 characters. According to the structure: horizontal and vertical movement of the image.
3. Graphic mode. Text may contain figures.

The above features facilitate a simple "full-screen editor" on microcomputer /under CP/M/.

To support data management services, text files may contain data fields. This gives new possibilities:

4. Allocation of data fields in text files /can be erased or re-allocated/.
5. Assigning type to data field. Basic types: numeric, alphabetic, alphanumeric or "any". The user can define special types /e.g. first byte of field is "+", "-" or space/.
6. Assigning initial value to data fields.
7. Data fields of a text file can be connected to other systems by using the interface provided by the editor. The interface is a free-format text optionally attached to any data field, its format is defined by the system using it /e.g. data base management system/.
8. Display and modification of the field attributes

/type, initial value, interface/.

Utilizing data fields, the editor can serve several aims:

- Data base input-output.
- Report generating.
- Easy-to-survey description of complex systems /the system parameters are the data fields/.

We wish to emphasize that most of the form editor commands are realized by function keys, so its operation is easy and very fast. There is no practical possibility to make severe mistakes. Therefore unexperienced users can utilize it, e.g. a data base input/output form can be edited in some minutes.

To utilize the form files stored on floppy disks, a form manager has been developed /to display forms and fill in data fields from the keyboard or from data base/. It is not a program but a collection of procedures that can support e.g. a DBMS.

The form management procedures of Micro-SHIVA are the following:

1. Loading of form into the memory.
2. Display and movement of form on the screen.
3. Cursor movement /usual modes and field access/.
4. Data field filling. Only the data fields can be written, other areas of the screen are not accessible. Data are checked immediately and the error is signaled if the syntax was violated.

. . The interface defined at form editing gives the possibility to perform semantic analysis, to store the data or to display derivated data. The fields are accessible

by names that are part of the interface.

3. Applications

We built up a simple data base system using the form editor and manager. The interface of the data fields is a four-letter identifier. The DBMS performs the following functions:

1. Data base generating /by filling a form/.
2. Data base modification /modification of the above form, including data insertion or deletion/.
3. Query by keys /max. 10 keys/.
4. Data display, storage, modification, deletion by forms.
5. Using multiple forms for one data base.

Up to now, our experiences show that unexperienced users welcomed the system when it was utilized in nursing systems for processing large forms and maintaining the data.

4. Bibliography

P. Kerékfy: GENERA: A Program Generator System; Progress in Cybernetics and Systems Research, Vol. XI., Trappl, Findler, Horn /eds./, Hemisphere /Washington/, 1982, pp. 117-122.

P. Kerékfy, A. Krámli, M. Ruda: SIS79/GENERA Statistical Information System; Progress in Cybernetics and Systems Research, Vol. XI., Trappl, Findler, Horn /eds./, Hemisphere /Washington/, 1982, pp. 123-128.

P. Kerékfy, M. Ruda: Program Optimization and Manipulation User Level; Cybernetics and Systems Research, R. Trappl /ed./, North-Holland /Amsterdam/, 1982, pp. 797-802.

P. Kerékfy, M. Ruda: Microcomputer-Based Medical Information Systems; MEDINFO 83 The 4th World Conference on Medical Informatics, Amsterdam, 1983.

P. Kerékfy, M. Ruda: Medical Information Systems on Desktop Computers; Conference on Computer-Based Information Servicing, Varna, 1983.

A. Krámli, M. Ruda, M. Csukás, M. Galambos: Large Sample Size Statistical Information Systems for HwB; Data Analysis and Informatics, E. Diday /ed./, North-Holland /Amsterdam/, 198o, pp. 457-462.

MTA SZTAKI Tanulmányok 183/1986 pp 157-175

M A D A M

MULTI LEVEL ACCESSIBLE DATA MANAGEMENT TOOLS

O. KISS, A. SZILLÉRY, G. SZÁDECZKY

Computer and Automation Institute
Hungarian Academy of Sciences
Budapest, Hungary

Abstract

A new approach is presented to the general challange of large mass data management in contrast to the traditional approach, the application of universal commercial database management systems. We introduce a flexibly configurable set of techniques, tools, modules realizing sets of well defined, efficient services. These are structured in logical levels similarly as in the phylosophy applied in the ISO OSI reference model. Users may access any of these interfaces, and are allowed to link only those parts of modules needed by the specific application.

1. Introduction

The database management tools introduced here can be characterized by the following analogy.

A construction set contains prefabricated elements for making the basement, the walls, the roof of a house, etc. Choosing the basement from the set, placing the elements of walls onto the previously selected basement elements and

putting the roof onto this construction a given house can be completed much faster than in the case when the elements are also made by the constructors.

The user of such a construction set only completes the house, the basic tools and means are available for him. It is clear though, that at the same time the constructors creativity is restricted by the ready tools. He can not build whatever wonderful house, only that one for which the basic tools are given.

The set of data base management tools provides at various levels /I/O level, file allocation, record management, etc./ tools to be selected by the user according to his requirements to integrate into his task.

This approach can be considered as following the main line of standardization works which develop architectural standard frames by deviding the problem into components. Well known results of standardization process are the ISO OSI reference model [1] in the field of computer communication or the proposal on architecture for data base management standard [2].

There are lots of reasons of this standardization efforts on both of the vendors' and users' sides. We do not intend to deal with them, only reference to our example about building houses.

The tools described here concern only the lower levels of data base management systems, the data base file system. The reasons for focusing the low-level end of the data base problem are the followings:

/i/ People might disagree what data model is the best, but a well structured data base file system is applicable for each data model implementation.

/ii/ Because most application programs have similar requirements regarding the kinds of data they use, the ground-

up method results in programmers writing the same kinds of code. What is needed therefore is a set of data management tools.

/iii/ Most applications need only a simple data base file system and not a large, complete and complicated data base management system.

We should like to contribute by this work - similarly to [1] and [2] - to the standardization efforts on the levels ranging from the file system level of operating systems till the data management systems.

Our goal is to outline an architecture for this field.

The tools of a file system level represent alternative possibilities /e.g. at the level of record management B-tree, dynamic hashing, etc./. These levels range from the operating system levels /I/O management/ to the more sophisticated record managements. The tools of a level are based on the tools of a lower level.

The interfaces between levels are designed to hide as much design and implementation decision as possible. This principle makes it possible, that a higher level tool can be implemented by different lower level tools and during the tuning a less effective solution can be changed by another more effective one.

The levels are connected to higher levels through functionally equivalent interfaces. The data management mechanism of a certain task can be configurated easily by designating one tool from each level and linking them into a module.

We should like to emphasize, that the data base management tools are not intended for end users, however some tools of higher levels may be used to solve simpler data management problems. Our goal is to provide means that can be applicable in a wide variety of problems and can be used

as the basis of varied, complicated and sophisticated tasks.

First we plan an experimental realization of the tools to be described here. The purpose of this implementation will be to check the correctness of the structure and to get convinced about the feasibility of all these.

Later certain levels may change and may get richer by new tools.

2. The environment

A lot of applications running in UNIX environment claim to means of data management. Though the UNIX operating system was not designed for data management but as a comfortable environment of program development the need to solve the data management is urgent.

We mention here only a few problems concerning the data management within UNIX/[3], [4]/.

/a/ The UNIX does not provide the continuous placement of a file on a disk.

b/ The paging made by UNIX together with the own paging of a data base management system means double paging resulting undesirable cost increase.

c/ The I/O operations of UNIX are materialized not directly but through the own page buffer of UNIX. The UNIX paging can not be influenced from a user program, for the delayed write the user does not know whether what he has written onto the disk is there or not.

d/ The UNIX does not provide tools for reconstructing the previous content of a file after hardware or software demage.

e/ There are no possibilities for the control of concurrent accesses.

The majority of the above problems must be solved by the data base management itself. But b/ and c/ suggest, that instead of the UNIX-V7 file system a new, modified one being more suitable for data management tasks must be applied [4].

At this moment we do not know and do not want to decide this question. The levels and interfaces of data management tasks are designed to be based either onto the UNIX file management or they can be fit to the UNIX as a special data base management organization. In the implementation first we choose the easier solution, i.e. we build on the UNIX file system. After the experiments with the UNIX file system, if that is desirable for efficiency,we have to modify the UNIX file system.

Later in this article we hint at those tools and levels which we think to be realized under the user interface of the operating system.

In the following section we describe the proposed levels and tools in detail.

3. Levels and tools of data base management

The levels of data base management, the tools within them and the interrelationships among levels are depicted in fig. 1. /The arrows represent the direction of usage./

We describe the different levels separately. In the description we emphasize not the details but the functions of levels and the important characteristics of tools arosen as alternatives within a level. The levels and tools described here will be judged after the experimental implementation i.e. they are not fixed, they can be changed and enriched by the experiences.

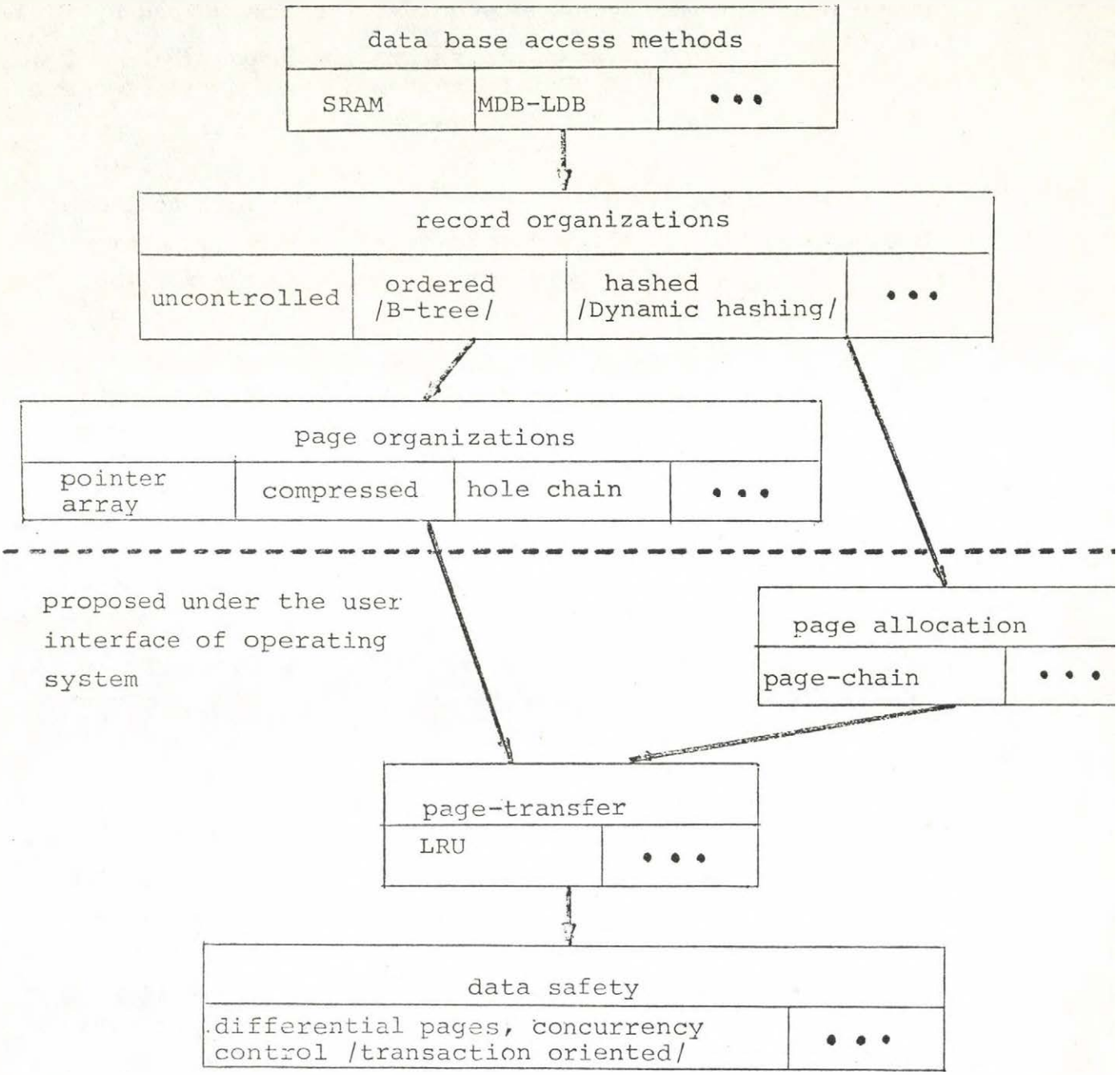


fig.1.

Levels and tools of data base management

3.1. The data base

In the followings we give a formulation suitable for us of the terminology "data base".

The data base is a given size storage area having numerated physical blocks fit to the storage media available for the data base management mechanism.

The storage area is devided into two separate parts, namely administrative blocks and data blocks /see fig. 2./.

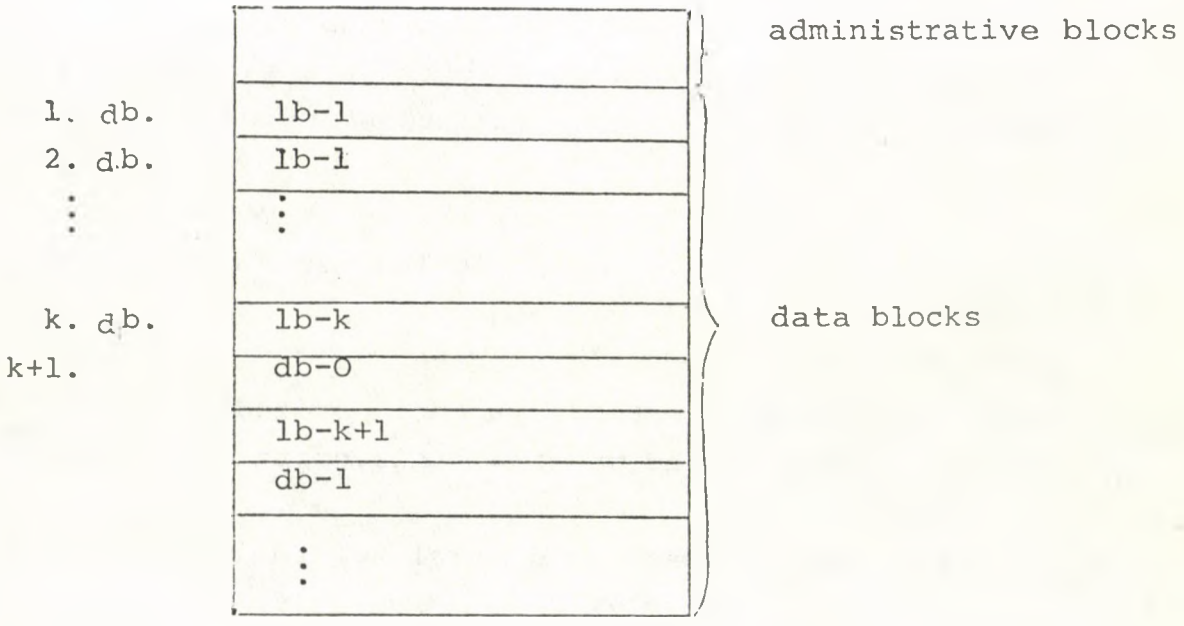


fig. 2.
The structure of data base

Administrative blocks are used to store the information controlling the data safety and concurrent access. /Detailed description of these blocks is given later./

The area of data blocks is devided furthermore into two not continuous parts, into logical blocks and differential blocks.

Logical blocks /in other words pages of the data base/ contain data structures of the data base while differential blocks serve for the storage of duplicates of logical blocks referred to by update transactions. When an update transaction is over the logical blocks referred to within the transaction become differential blocks for later usage and the previously differential blocks connected to the transaction become logical blocks. /The reason of non continuous logical and differential parts./

An important problem with the above method is that it does not provide to be the logical order of data base pages the same as their physical order.

The 0-th logical block of the data base contains a directory registering the structures stored in the data base /and possibly the organizations containing schema informations/.

When the data base is initialized the directory is empty. The usage of directory will be shown at the description of page allocation.

3.2. Data safety mechanism

The essence of data safety mechanism suggested by us is the following. The non consistent duplicates of pages referred to by update transactions are stored within differential blocks until the end of transaction. When the transaction has finished the already consistent duplicate becomes part of the data base

containing structures and the logical block containing the old version becomes differential block /[5]/.

The above mechanism works under the control of tables stored within the adminsitrative blocks. There are two tables within the administrative blocks:

- control tables of pages
- map of data blocks.

The structure of the tables is the following:

a/ Control tables of pages

For each page it contains one entry. The structure of an entry is depicted in fig. 3.

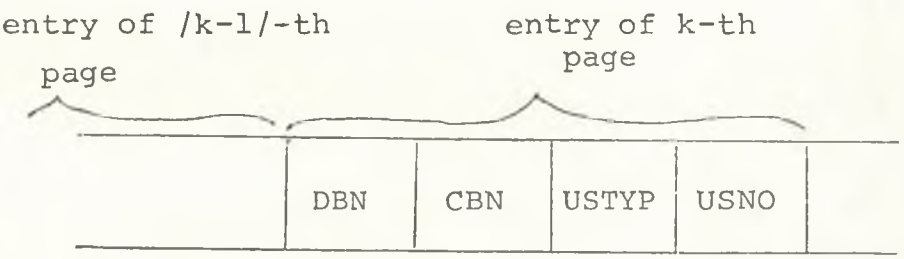


fig. 3.
Control tables of pages

The explanation of the abbreviations

DBN - the sequence member of data block being a differential block and containing the inconsistent state of page /during the update transaction/.

CBN - the sequence number of data block being a logical block and containing the consistent state of page /before or after the transaction/.

USTYP, USNO - used only by the mechanism controlling the concurrent accesses /we give their meaning in the following section/.

b/ Map of data blocks

Gives for each data block of the data base whether

- it is used /logical block or currently used differential block/
- or not /not used differential block/.

For storage of CBN's the data base always has consistent state. After an update transaction CBN=DBN's will occur in the entries of the referred pages.

The safety of administrative blocks can be solved by duplication and by setting a validity sign bit in the following way: when each update transaction has finished the validity sign bit points to the valid duplicate of administrative blocks.

3.3. Control of concurrent accesses

Here we deal only with a simple, transaction oriented rollback mechanism applying lock at physical level. Later, after the first realization of data base management tools other more effective methods can be chosen.

The mechanism proposed by us is based on the following simple principles.

a/ A /update or query/ transaction locks the referred pages until the end of transaction.

b/ A page already locked by an update transaction can not be locked by a subsequent transaction.

c/ If an update transaction needs access to a page already locked by another /update or query/ transaction, then the update transaction rolls backward, i.e. frees the pages /and possibly the differential blocks already connected to it/ and starts again /after delay/.

d/ If a query transaction needs access to a page already locked by an update transaction, then the query transaction rolls backward and starts again /after delay/.

Using this simple principles no deadlock may occur. Lock of pages is performed by setting the fields USNO and USTYP in the control table of pages.

USTYP - type of lock /query, update, not locked/
USNO - number of transactions locking the page /in case of update transaction equals 1/.

3.4 Page transfer

The page transfer is the mechanism providing the physical I/O of pages in the data base. The I/O is realized by a page buffer. For page transfer we propose the LRU algorithm regarded as generally good one [6] .

However we note that in certain cases not the LRU is the most suitable strategy [4].

The uniform interface of page transfer - the algorithm can be changed by changing a parameter of the interface - makes it possible that later this level can be enriched by other algorithms.

The page transfer is based on the level of data safety and concurrency control but this fact is hidden by the very simple interface.

3.5 Page allocation

The principle of page allocation can be formulated in the following way.

For the internal organizations and structures pages are allocated from the available logical blocks of data base or respectively empty pages are reinserted into that.

The principle above is very similar to the file allocation function of operating systems. E.g. the UNIX operating system administers the available storage area, creates and deletes file by allocating storage areas for files or respectively freeing storage areas occupied by the deleted files.

The above similarity poses the following possibilities

a/ The data base is an area managed by the operating system.

b/ The data base is not under the control of the operating system. Though until a certain level the operating system can manage the data base /e.g. in case of UNIX until the level of insertion into directory/ however other file management services belonging to the operating system or missing from it /e.g. data safety, page transfer, structure management/ can be replaced by solutions functionally equivalent to the former ones but much more sharpened for the special requirements of data base management.

Both of the above alternatives contain advantages and disadvantages. The choice of a/ makes the implementation much more simpler but the duplicates of certain functions belonging to the operating system and also to the data base management system /e.g. page transfer, page allocation/ and the data safety and concurrency control mechanism being strange for the UNIX operating system may cause substantial cost increase. The b/ alternative causes the modificaton of the UNIX file system.

In the experimental version we choose the simpler a/ alternative.

The page allocation function will be performed in the following way.

At the creation of a structure /record organization/ a directory entry containing the type and name of structure will be filled. The directory entry refers to the header page/s/ of the structure. The header page/s/ allocated at the creation of structure contains administrative information concerning the structure and depending on the type of structure /e.g. in case of sequential structure the sequence numbers of pages belonging to it - see fig. 4./.

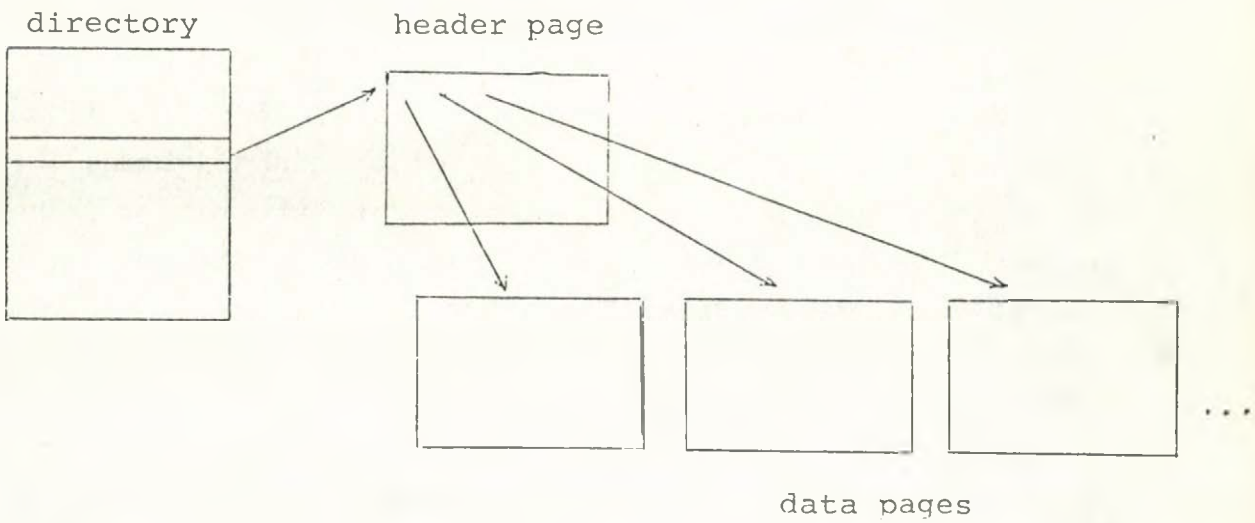


fig. 4.
Sequential structure administration

When a structure requires a new page the allocation will be performed using a chain of free pages. A new page will be connected to the structure by administering it in the structure header page /if it is necessary/. When a page becomes empty it will be released and conncected again to

the free page chain.

In case of the deletion of a structure the header page is released and the directory entry referring to it is cleared.

The uniform page allocation interface can be controlled by parameters.

3.6 Page organizations

The data on pages can be organized using different methods. /Here we use "data" terminology instead of "record", since at this level the components of records are not known, they have only one characteristic, the size in bytes./

We propose the following page organization methods:

a/ compressed

b/ hole chain

c/ pointer array.

The above methods /see e.g. in [6] / differ in their data placement strategy and deletion mechanism. The different organizations are connected to the higher levels of data base management tools through different but functionally equivalent interfaces.

3.7 Record organizations

By record organization we mean the grouping of records into a given type structure /logical file/. The level of record organizations is based upon two lower levels, the levels of page organization and page allocation. A record organization, i.e. the structure type determines certain tools of these levels.

Here we deal with three record organizations. These are the

- uncontrolled
- ordered /B-tree/ and
- randomly accessible unordered /dynamic hashing/

structure types.

Of course the three solutions above do not exhaust the available record organizations. It is also clear, that the individual structure types can be implemented by other methods /e.g. the order by chaining, the random access by other kinds of hashing/. Our choice does not mean the record organizations not to be enriched by other methods later.

Before the discussion of structure types we must remark the followings:

a/ At this level of data management tools we mean by record a group of data having content. The record structure is not important for us, merely the key field's size and position within the record is interesting.

b/ The records to be organized into a given structure are formally homogeneous, i.e. for each record occurrence the size and position of key field or fields /if exist/ are the same. Here we do not deal with unhomogeneous structures /e.g. CODASYL sets/ because these can be built utilizing the homogeneous ones.

3.7.1 Uncontrolled record management

In the experimental realization the pages belonging to an uncontrolled structure will be organized by pointer arrays. We should like to remark, that at this level this is an arbitrary decision. For uncontrolled record management the compressed and hole chain organizations also suit /requirements arising later may necessitate the realization of these ones/. The reason of choosing pointer array solution is the implemantation

of a tool at a higher level in the experimental version.

It is necessary to speak about the name of this record management. The uncontrolled record management is in fact a sequential record organization. The sequential placement mechanism is disturbed only by the deletions and so it becomes uncontrolled for the user.

The essence of this record management.

A new record will be placed onto the first page belonging to the structure and having enough free room for the record. The mechanism of deletion is determined by the pointer array page organization.

3.7.2. Ordered / B-tree/ record management

The pages belonging to ordered structures are compressed. Here we do not discuss the essence of B-trees. Detailed description can be found in [6].

3.7.3 Hashed /dynamic hashing/ record management

In the experimental version the pages belonging to hashed structures will be organized using hole chains. Because the various page organizations have functionally equivalent interfaces, later this choice can be replaced by either compressed or pointer array organizations.

The exact description of the algorithms of dynamic hashing can be found in [7].

3.8 Data base access methods

The data base access methods represent the highest level of data base management tools discussed here.

We mean by a data base access method a coordinated,

intelligent data management mechanism of structures realized by different record organization methods.

The level of data base access methods represent a higher step of finer quality compared with the other levels discussed so far, since with the type definition /relation type or record type/ means and other "type oriented" manipulations provided at this level certain tools in themselves can be applied as simple less structured data base management system of user.

At this level we can speak about the fields of data records and automatic maintanance of one ore more indexes. A tool of this level can be driven by a table specifying the fields, how the keys are formed, what kind of access methods can be used for the indexes, etc.

We mention here two data base access methods, the logical data base handler of the MDB screen oriented data base management system [8] and the SRAM relational interface intended to be realized in the experimental version.

3.8.1 Logical data base handler of MDB

The MDB is a screen oriented data base management system with automatic schema management. Under the user interface it provides a mechanism serving for the storage of atoms and binary relations among them. For the functional description of the data base handler see [9]. It was implemented by means of B-tree and uncontrolled record organizations.

3.8.2 SRAM /Simple Relational Access Method/

The user can define relation types by giving the types of attributes and the value constraints belonging to them

/giving the schema/.

The rows of a relation type are stored by uncontrolled structure, however based on the indication of user B -trees can be organized for certain attributes. The records of structures organized by B-tree contain the attribute value and a pointer to the row in the uncontrolled structure /fig. 5./.

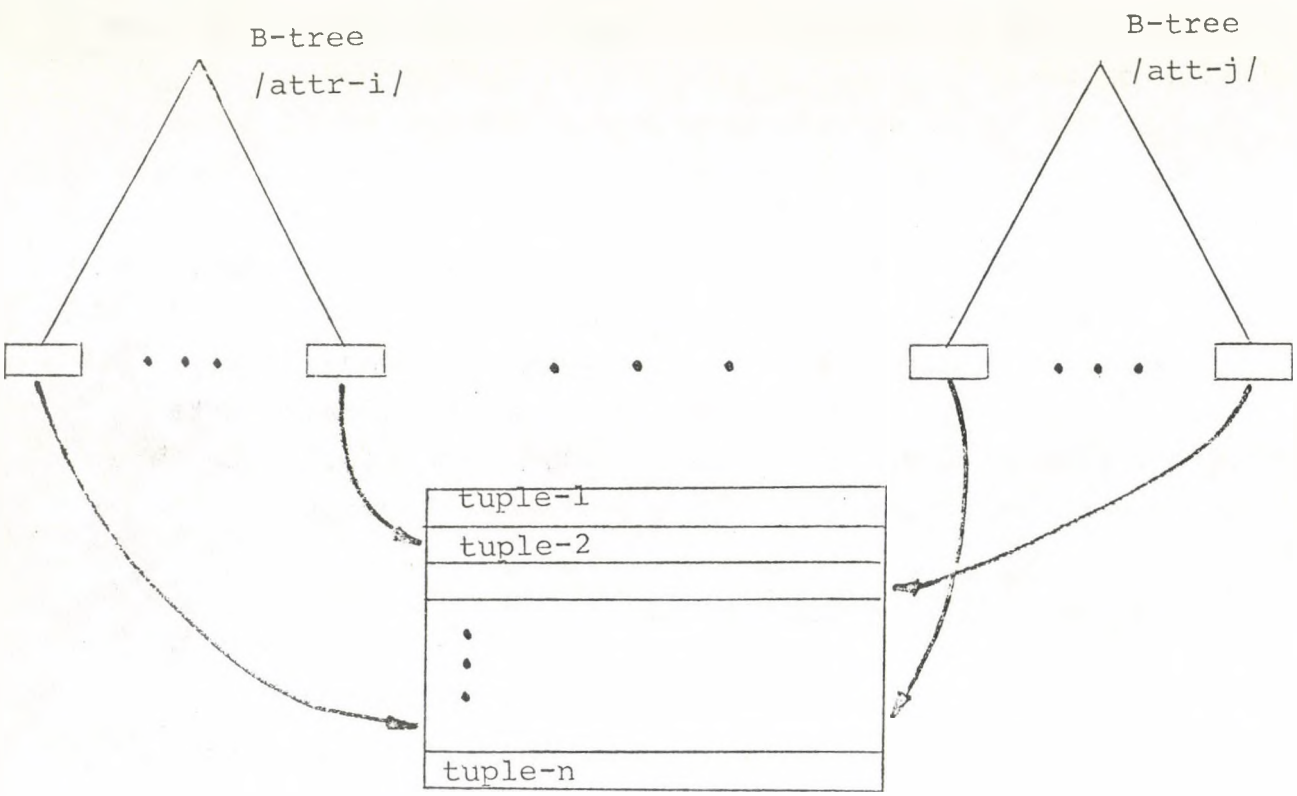


fig. 5.
The organization of SRAM

We note, that B-trees can also be organized later, i.e. for the rows of a relation already filled by data B -tree can be requested.

References

1. ISO "Computers and Information Systems-Open Systems Interconnection - Basic Reference Model", DIS 7498, 1981.

2. An architecture for database management standards ISO /TC 97/SC 5/WG 5 N 28, 1982.

3. D.M. Ritchie, The UNIX I/O system, UNIX Programmer's Manual, vol 2, 1979.

4. M.Stonebraker, Operating System Support for Data Base Management, CACM vol. 24, No.7. 1981.

5. M.F. Challis, Database Consistency and Integrity in a Multi-User Environment, in Databases: Improving usability and responsiveness, Academic Press, 1978.

6. J. Martin, Computer Data Base Organization, Prentice Hall, 1975.

7. P. Larson, Dynamic hashing, Bit 18, 1978.

8. E. Knuth, MDB Project proposal, MTA SZTAKI WP II/26, 1981. /internal material/

9. E. Knuth, MDB Machine independent specification, MTA SZTAKI 1983. /internal material/

MTA SZTAKI Tanulmányok 183/1986 pp 177-182

COMPUTER - AIDED DATABASE MANAGEMENT SYSTEM IN THE HERNAD "MARCIUS 15.MGTSZ" AGRICULTURAL COOPERATIVE A CASE STUDY

G. REMZSŐ

Technical University of Budapest,Computer Centre,
Hungary

1. The most important activities in the cooperative

The Hernad "Marcius 15. MGTSZ" Agricultural Cooperative is the cooperator of the HUNNIAHIBRID production association which was called to existence for the production of broiler-chicken. In this field of activity the co-operative has a close business link with the Dutch EUROHIBRID Company. In the process of the broiler production small estates have a high importance as most of the breeding takes place there.

The most important activities in the co-operative are:

animal breeding and processing
plant growing
fodder processing
management farming plots
complementary industrial activities
food processing
consulting members of the HUNNIAHIBRID

With respect to its monetary value the most important is the animal breeding (and poultry production, respectively) branch.

2. COMPUTERS IN THE CO-OPERATIVE

In the HERNAD "Marcius 15. MGTSZ" Agricultural Co-operative there are the following computer hardware systems:

IBM SERIES/1
INTEL 80 DATA COLLECTION SYSTEM
VIDEOTON EC 10/M (VIDEOPLEX)
IBM PC/XT.

In the centre of the computer network is the IBM SERIES/1 computer. This computer has the following hardware configuration:

PROCESSOR IBM 4955-F00 256 KByte
Timers
Floating point
Communication Indication Panel
Programmable Communication Features

I/O Expansion Unit 4959-A00

Diskette Magazine Unit IBM 27.8 MByte

Disk Subsystem 28 MByte

Display terminals IBM 4978-1 4 pcs

Matrix printers IBM 4974-1 3 pcs

Teletype Displays ORION ADP 2000 18 pcs.

TTY lines.

The configuration of the EC-10/M:

CPU 128 KByte

Magnetic tape units

Disk 20 MByte

Punched card reader

Line printer

The configuration of the IBM PC/XTs:

CPU 256 KByte

Winchester disk capacity

Floppy disk (360 KByte)

Matrix printer

IBM 8087 Math. Coprocessor.

3. Software Systems on the IBM SERIES/1

In the IBM SERIES/1 the HERNAD-INFO system was developed:

Collection of poultry production branch data;

Book-keeping system;

Farm accounting system;

Marketing system;

Stock management;

Optimal control of the fodder plant;

Production information (daily, weekly, monthly, yearly);

Information management of the HUNNIAHIBRID System.

4. The work of the VIDEOPLEX system

In the VIDEOPLEX there is a data preparing system for the other computers.

The large capacity line printer is connected to the IBM SERIES/1, too.

5. THE WORK OF THE INTEL 80 SYSTEM

This system is the part of the LINDHOLST Chicken Slaughtering Equipment. The collected data of this computer system is sending to the IBM SERIES/1.

6. ACTIVITIES IN THE IBM PC/XT COMPUTERS

We proposed to introduce a Consulting system for the HUNNIAHIBRID union, the name is HUNNIA.

Connecting with the other systems the HUNNIA gives the following data:

information about partners and contracts;
information for poultry breeding technology;
statistical data of the HUNNIAHIBRID system;
information for the optimal animal's production;
collected information of the HUNNIAHIBRID system;
trace the egg production;
trace the hatching and the fodder processing data.

The HERNAD-INFO is developed and used, the HUNNIA is under development. The above computer aided database management system is a powerful instrument in the management of the work in the co-operative.

REFERENCES

1. DATE,C.J., An Introduction to Database Management Systems
Addison-Wesley Publishing Company
1977.

2. BENASTEAU,D, Comment creer votre propre banque de donnees, et la reussir
L'Usine Nouvelle
N° 21. 1985.

3. MILL, J., An executive tool or toy?
Computing the Magazine
N° 4. 1985.

MTA SZTAKI Tanulmányok 183/1986 pp 183-187

RELATIONAL DATABASE MANAGEMENT SYSTEMS FOR MICROCOMPUTERS

T. REMZSŐ

Comp. and Autom. Inst. Hung. Acad. Sci.
Hungary

The microcomputer relational database management systems now present the DP manager with a threat and an opportunity. If the micro Relational DBMS is ignored in favor of a more traditional, large systems approach, the users are apt to grow restive. They are aware that new products can do many of the things an on-line system can, and they know that developing large systems takes time.

Even if the DP manager agrees to go the micro area, problems may occur. Development methods appropriate for large systems won't work here! Slipshood analysis will produce muddled systems that soon become unmaintainable. And lack of integration work will render the system's useful life very short.

Relational database systems for microcomputers are a way to give users the functions they have been waiting for. Properly implemented, they can help the DP manager cut through the applications backlog and move his organization into effective distributed computing.

A good way to manage the intoduction of microcomputer RDBMSs into the corporation is to form a special team to handle the new technology. The team needs the support of top management because it will have to assign priorities to jobs for different departments, and also recommend operational changes. This team should be assembled from representatives of the DP and user departments, plus outside consultants.

To purchase hardware and software, the team needs a *budget*, nearly the price of a minicomputer. Because the team will have to move from department, *portable facilities* are important.

Using an RDBMS it is possible to have a life cycle with the following phases:

1. INCEPTION

   Covers the business problem, possible solutions, and the selection of the most cost-effective solutions.

2. ANALYSIS AND MODELLING

   Delivers a model of what the current systems do, with emphasis on the ralational concepts.

3. EQUIPMENT SELECTION

   Covers the selection of DBMS and hardware.

4. IMPLEMENTATION AND TESTING

   Installation the equipment and put the ralational model onto the RDBMS.

5. TRAINING

   Training users and giving them responsibility for the new system.

6. EVOLUTION AND INTEGRATION

   Maintenance of the system through its operational life.

The important RDBMS property here is the distinction between function and system interior. In other words, *the distinction between what the system does and how it does it.* The system's top layer follows the well-behaved relational model, making it easy to understand and use. The bottom layer is hidden from view; it contains all the nonlinear physical structures required for the efficient handling of disks. The top layer provides external appearances of logical data structures, which are called virtual views. It is this separation of logical and physical considerations that makes the special RDBMS life cycle possible.

An RDBMS hides all implementation details, therefore it becomes possible to take the relational model of the current manual systems and simply tell the RDBMS what that model is. Once this model becomes known to the system, the RDBMS gains the properties of that model and takes on its behaviour. Thus the new computerized system is quickly created. This is how the low-level design and implementation of that design are bypassed. It is function, not design, that is implemented using RDBMS.

REFERENCES

1. Office Automation and DP

   Xephon User Survey

   1985.

2. Cotton,I., Technologies for local area computer networks

   Computer Networks

   Vol. 4. N° 5. 1980. pp. 197-209.

3. Introductin to local area networks

   Digital Equipment Corporation

   1982.

MTA SZTAKI Tanulmányok 183/1986 pp 189-196

A GENERAL-PURPOSE DATA MANAGEMENT SYSTEM

T. LENGYEL and *Z. TÓTH*

Computer and Automation Institute
Hungarian Academy of Sciences
Budapest, Hungary

ABSTRACT. MEDAC is designed to create and manage general-purpose data base systems for applications requiring search according to several different keys and user defined printing forms. MEDAC handles several independent data bases each one having a main and several auxiliary data files, linked by pointers. MEDAC is implemented on a Z80 based microcomputer of 64 Kbyte with CP/M operating system.

INTRODUCTION

We describe a data base management system whose name 'MEDAC' is an abbreviation of MEdical DAta Collector which shows the origin of the system: it was designed to facilitate and aid the labour of medical workers.

The main aim was to provide software utilities for keeping records and registers in health care applications emerging in small clinical departments, laboratories and the every day medical practice.

Another possible use of the system is the incorporation into a complex medical system, i. e. connecting the microcomputer with medical equipments and collecting the mesurement values.

Despite general-purpose features of the system its usage needs neither practice in computer technique nor special background.

HARDWARE ENVIRONMENT

MEDAC is developed for MOD-81, a Z80 based microcomputer of 64 Kbyte memory. 8" flexible disc drives are used as mass storage devices.

MEDAC runs under CP/M operating system. The programs are written in Pascal MT+ and Z80 assembly.

Most of the programs do not utilize special features of the MOD-81 microcomputer so the system is portable - with some alterations - to any similar computer.

ARCHITECTURE OF PROGRAMS

The system has a complex structure combining different program linking technics. Having relatively little memory at disposal, only certain parts of the system programs can be loaded into the memory at the same time. For this reason the system consists of several modules which are linked in different ways. Some comparatively small functions are supported by overlay modules while the large and self-dependent routines are connected by chaining technique.

Modules of the system are written with care so they are self-contained, can be used - and are really used - in various program systems.

DATA HANDLING

There are data files of two different types in MEDAC: the main file and the auxiliary files. The files are connected by pointers which point from components of records of the main file to records of auxiliary files.

The role of the auxiliary files is providing additional information about the records of the main file. This approach has the advantage that the user is able to store repetitive information in the auxiliary files. This also reduces the space requirement so this technique is worth using if there is a great need for economical use of memory.

SPECIAL FILES

The system uses three file types which differ substantially from data files. B-tree files are built dinamically by the system and are physically stored as index files on the disk. The upper levels of the currently used B-tree are moved to the memory while other parts are read from disk if new branches of the the tree are needed during the search. The report description files which support user-defined printing forms are text files and code the information in a special way. Description files of micro-SHIVA form editor [5] are stored in text files of a special form.

MAIN FEATURES OF MEDAC

The usual functions of similar interactive databases for microcomputers can be found in MEDAC for adding, deleting or modifying data. Besides the data handling there are two main features of the system. One of them is that it supports search according several keys. This is realized by organizing B-trees for ordering records according to different keys. The other special feature is enabling different user-defined printing and displaying forms. This is provided by a special report utility described later in details and micro-SHIVA form editor which was already mentioned.

DATA ACCESS

The data records can be accessed by record numbers or keys using B-trees. If a record is already reached relative search commands can be used as well: the next or the prior record according to

the ordering or all other records with the same key can be requested.

ADDITIONAL SERVICES

Besides the main functions MEDAC provides some additional services too. A lot of helping routines are built into the system providing a short explanation easy to understand even for unexperienced users as well.

An other additional service helps balancing B-trees. After a long use B-trees may become less and less balanced which increases the access time, especially in case of deleting numerous records. This handicap can be eliminated with a utility reorganizing B-trees.

Other additional functions that MEDAC provides include listing report descriptions with their main parameters, listing B-trees, making back up of the system etc.

REPORT MAKING

During the system design one of the main ideas was making possible printing information stored in the data base in a free, user defined form. Practical experience shows that the printouts are very important and their suitable form makes them much easier to survey and fitter to use.

Another important aim was to enable the user to make computations during report making with the data involved and to print the results of these computations together with the original data of the data base. It may be necessary to compute and print the mean value, the sum or the variance of the data of certain fields. Making possible these calculations seemed to be really important in practice especially in case of reports that give an overall picture on some aspects of a big amount of data.

During the report making one can take advantage of other features of the system as well. Any report can be combined with any sorting of data. This is very useful in case of listings according to some respect. E. g. with the same report description one can generate a list of pharmacies in alphabetic order or a list according to their prices using the appropriate sorting of data. Another important point that the user is able to prescribe conditions for the fields of records that are admitted to the report.

REPORT DESRIPTIONS

As it was already mentioned the system makes special files called report description files that contain information needed for making reports. A report description consists of different lists and calculation commands. The entities to be printed are gathered in lists which may contain data base items, i. e. component of data records and variables, i.e. entities whose values are computed from data items.

Lists of another kind define the printing format using different control characters. These lists consist of various control characters that define the character set to be used in different parts of the report, the position of the items on the pages etc. These lists also contain fixed character sequences that must be printed during report making. The computations take place by means of variables defined by the user at the beginning of the report. The rules of their calculation is given at the same time. The sequence and timing of their actual computation are described in lists of another special kind. As it was already mentioned, values of the variables can be printed just as those of data base items. However, the calculation always happens with the actual values of the items and the other variables, it can take place several times during the report making and timing of the calculation is independent of the printing.

There are also special variables, not mentioned before, called initialized variables. The value of these variables can be given each time by the user when they appear during the preparation of the report. This makes possible for the user printing comments and supplements into the report and giving a nonzero initial value for certain computations if necessary.

The report description is controlled by a special user friendly utility of the system . This utility controls the user, checks the consistence of the different parts of the description, the consistency of the whole description with the data base structure. A lot of helping messages are given in this routine which explain how the control characters are to be used, give instructions to the user etc. If a command or answer of the user is syntactically or semantically incorrect or would lead to an inconsistent state of the whole description then the answer is refused by the system, an error message is given with an exact explanation of the cause of the error and the system is put back to the status before the incorrect command. E. g. the system refuses putting into a list entries incompatible with prescriptions concerning these entries in other lists that already exist.

ARCHITECTURE OF REPORTS

The whole report consists of four parts. The introduction gives general information on the report, it usually contains the name of the report and the date of report making. The page header is printed on the top of pages and as a rule contains page number or the report name. The items of the data base are printed in the main part of the report, while the final part usually contains the result of the computation concerning the whole data amount involved in the report. All parts are optional, of course, any of them can be omitted.

APPLICATIONS

As it has been already mentioned, the system was designed for

health care usage. We list first some concrete applications in this field. In hospitals the anamnesis of the patients can be efficiently stored and managed by means of the system. Report utility is suitable for printing final reports or statistics on medicinal treatment, therapeutics etc. There are applications where the administration of consumption of medicines is supported and reported by the system.

MicroSHIVA enables filling up and modifying medical forms in a free mode. The system makes possible handling data in a convenient form for the members of a medical staff (physicians, sisters, laboratory assistants etc.) according to their rank and position.

Being a general-purpose system, medical practice is not the only field of possible applications of MEDAC of course. It can be used in particular in fields requiring search according to different keys, intelligent report making and form handling routines. Similar systems are applied and MEDAC may be used in business and office applications like letter recording, employment or store recording and many other similar fields.

REFERENCES

1. L. Hannák, K. Kovács and T. Lengyel. On applicability of personal computers in data storage of hospitals and in other fields of health care. In "Applications of computers and cybernetic methods in medicine and biology," 10th Neumann conference, Szeged (1980) pp. 77-84 (in Hungarian).

2. K. Kovács. Personal Data Management System on microcomputers. In "Proc. of 2nd Neumann Conference," Székesfehérvár (1983) pp. 73-77 (in Hungarian).

3. P. Bakonyi, A. Békéssy, J. Demetrovics, P. Kerékfy and M. Ruda. A microcomputer based decision support system for health care organizations. In "A bridge between control science and technology," vol. 11 pp. 85-92. IFAC 9th World Congress, Budapest (1984).

4. D. E. Knuth. The art of computer programming, vol. III: Sorting and searching. Addison-Wesley, Boston (1973).

5. P. Kerékfy and M. Ruda. MicroSHIVA user friendly information system development in medical application. In "Lecture Notes in Medical Informatics," 24th volume, D. A. B. Lindberg and P. L. Reichetz (editors). Medical Informatics Europe 84, Brussels (1984) pp. 235-239.

1985-BEN MEGJELENTEK:

166/1985 Radó Péter: Információs rendszerek számitógépes tervezése

167/1985 Studies in Applied Stochastic Programming I.
Szerkesztette: Prékopa András /utánnyomás/

168/1985 Bcszörményi László - Kovács László - Martos Balázs - Szabó Miklós: LILIPUTH

169/1985 Horváth Mátyás: Alkatrészgyártási folyamatok automatizált tervezése

170/1985 Márkus Gábor: Algoritmus mátrix alapu logaritmus kiszámitására kriptográfiai alkalmazásokkal

171/1985 Tamás Várady: Integration of free-form surfaces into a volumetric modeller

172/1985 Reviczky János: A számitógépes grafika terület-kitöltő algoritmusai.

173/1985 Kacsukné Bruckner Livia: Mozgáspálya generálás bonyolult geometriáju felületek 2 1/2D-s NC megmunkálásához

174/1985 Bolla Marianna: Mátrixok spektrálfelbontásának és szinguláris felbontásának módszerei

175/1985 Hannák László, Radó Péter: Adatmodellek, adatbázis-filozófiák

176/1985 Számitógépes képfeldolgozási és alakfelismerési kutatók találkozója.
Szerkesztette: Csetverikov Dmitirj,
Főglein János és Solt Péter

177/1985 Gyárfás András: Problems from the world surrounding perfect graphs

178/1985 PUBLIKÁCIÓK'84
Szerkesztette: Petróczy Judit