tanulmányok 213/1988

MTA Számítástechnikai és Automatizálási Kutató Intézet Budapest

Исследовательский Институт Вычислительной Техники и Автоматизации
Венгерской Академии Наук
Computer and Automation Institute , Hungarian Academy of Sciences

ИССЛЕДОВАНИЕ И РАЗРАБОТКА ИНТЕГРИРОВАННЫХ СИСТЕМ ОБРАБОТКИ ИНФОРМАЦИИ (НА ПРИМЕРЕ СИСТЕМЫ СТРАХОВАНИЯ)

Ремжё Тибор

Tanulmányok 213/1988
Studies 213/1988

A kiadásért felelős:

DR. KEVICZKY LÁSZLÓ

Főosztályvezető:

DEMETROVICS JÁNOS

Настоящая работа поддерживается при OTKA No 1066

ISBN 963 311 255 9
ISSN 0324-2951

ОГЛАВЛЕНИЕ

стр.

Введение

Развития средств, методов и технологий обработки информации претерпели многие изменения за долгую историю развития человечества. С древних времен шло постепенное накопление информации в виде знаний, опыта, но на рубеже $40^{\underline{х}}$ годов $XX^{\underline{го}}$ века произошёл так называемый информационный взрыв, когда количество и качество информационных потоков стало неуправляемым и человечество начало "захлебываться" в поступающих данных. Возникновение и развитие вычислительной техники вселило надежды, что найден инструмент, который позволит справиться с объемами информации и, более того, заключить её в различные, технологически обоснованные процессы управления, что, по замыслу учёных должно было резко улучшить управление всеми областями общественного бытия. Появились работы Винера, Глушкова, обосновывающие возможность и необходимость создания локальных и глобальных систем обработки информации, концепции автоматизированных информационных систем (АИС) и их воплощение в автоматизированных системах управления (АСУ), региональных автоматизированных системах управления (РАСУ), общегосудрственных автоматизированных системах (ОГАС) и др. Однако, несмотря на достаточно глубокое теоретическое обоснование, АСУ регионального и государственного уровня созданы не были. Более того, ни одной стране, включая таких "китов автоматизации" как США и Япония не удалось этого сделать и они понесли значительные материальные и моральные потери. Объяснение этого следует искать в недооценке следующих факторов:

1. Чем многозвеннее цепь управления, чем больше в ней информационных входов, тем быстрее старится информация. И в зависимости от информационной технологи, совершенства средств ВТ и пропускной способности каналов рано или поздно наступает момент, когда информация устаревает не успев попасть в места хранения и использования.

2. Создание глобальных систем управления требует не только и не столько совершенства средств ВТ, а коренного пересмотра самого управления, его упрощения и приведения в соответсвие с требованием времени. В противном случае автоматизация беспорядка вызывает беспорядок в квадрате.

3. Человек, включённый как коммутационное и управляющее звено в систему управления, оказался к этому не готов.

4. На всех этапах обработки информации: ввод, обработка, хранение, выдача - существуют "тонкие места", которые в совокупности могут сделать систему обработки неэффективной.

5. Объект управления - общество оказался слишком сложным и плохо поддающимся математическому описанию.

Крах попытох создания управленчиских АСУ в 70^{е} годы привёл к переосмыслению понятия информационной технологии и заставил перейти к выработке приемлемых локальных концепций автоматизированных систем. Одновременно шло переосмысление задач управления , разработка современных методов руководства, совершенствование компьютеров (скорость, объёмы памяти, улучшение

интерфейса с пользователем и др.) и средств связи.

В конце $70^{\underline{x}}$ годов были созданы системы управления и информирования следующих типов:

1. Информирование по сферам деятельности (библиотечные, медицинские, военные и др. системы)

2. Системы принятия решений локального типа (экспертные системы, системы управления предприятиями и др.)

3. Локальные сети.

Создались предпосылки для объединения локальных сетей в глобальные.

В это же время возникли и бурно развились персональные ЭВМ, а также системы и сети на их основе. Возникли системы VIDEOTEX-а, TELETEX-а, национальные страховые и биржевые системы.

Однако, создание и развитие таких систем не является постепенным переходом количества в качество. Существует целый ряд проблем, решение которых необходимо обеспечить сейчас, чтобы завтра локальные и глобальные распределённые сети получим, если не строго научное, то по крайней мере законченное методологическое обрамление. Тогда разработка автоматизированной распределённой системы управления перестанет быть уникальным действием и приобритёт облик тиражируемости по технологии.

Для этого необходимо:

1. Создать и исследовать информационные модели локальных и глобальных информационных систем.

2. Разработать типовые архитектуры таких систем.

3. Исследовать законы распределения информационных потоков в сетях.

4. Выработать рекомендации по созданию интелектуальных интерфейсов между человеком и информационными системами.

5. Решить вопросы обновления и актуальности информации.

Частичному решению этих и ряда других вопросов (в контексте создания системы страхования) и посвящена настоящая работа.

Новизна работы заключается в том, что в ней расмотрен новый подход к созданию распределённых систем обработки данных на основе создания информационной модели, выключающей диалоговые информационно-распознающие алгоритмы.

Актуальность работы заключается в том, что разработанная информационная модель и созданная на её основе архитектура системы страхования , а также технология ведения распределённой информации могут является основой для создания класса информационно - управляющих систем таких как : медицинские, системы резервирования, юридические и др.

Практическая ценность диссертационной работы заключается в том, что математические , технологические и методологические рекомендации воплотились в создание распределённой системы LATOR и разработке на её основе системы страхования BHP ABLAK. В настоящее время заключён договор с рядом западноевропейских фирм (в частности с фирмой OLIVETTI) на поставку этой системы, а также ведутся переговоры с СССР.

Диссертационная работа состоит из введения, трёх глав, заключения, списка использованной литературы и приложений.

Во введении кратко рассмотрена предыстория развития распределённых систем обработки информации, определен предмет исследования, а также обоснованы новизна, актуальность и практическая ценность работы.

В первой главе рассматриваются многоуровневые распределённые системы обработки, соотношение между интеграцией баз данных (БД) и функций обработки, технические и программные средства, на основе которых можно реализовать системы интегрированной обработки. Анализируется локальные сети и формулируется задача создания информационной модели, включающей диалоговые информационно - распознающие алгоритмы, и разработка на этой основе системы страхования.

Во второй главе определяется и изучается информационная модель распределённой обработки информации. Вводится понятие

диалоговых информационно-распознающих алгоритмов и производится их исследование в рамках интегрированных (по БД и функциям обработки) информационных сред. Производится конкретизация информационной модели до функциональной модели системы страхования. Анализируется интерфейс с пользователем системы страхования с точки зрения функционирования диалоговых информационно-распознающих алгоритмов. Функциональная модель системы страхования рассматривается как последовательно наращиваемая модель.

В третьей главе описывается архитектура реализация распределённой СУБД **LATOR** и создание на её основе системы страхования ВНР - **ABLAK**. В рассмотрение включен программно - технический интерфейс с пользователем. Описаны функции обработки информации разных уровней. Произведена оценка эффективности систем **LATOR** и **ABLAK**.

В заключении приведены основные результаты диссертационной работы и сделаны выводы по дальнейшему использованию этих результатов.

В списке основной использованной литературы приведены основные библиографические источники.

В приложении приведена справочная информация, необходимая для более полного понимания систем **LATOR** и **ABLAK**.

Основные результаты диссертационной работы докладывались на семинаре ИИВТА ВАН по проблеме "СУБД", на семинарах РГ 25 и РГ 26 КНВВТ в г. Киев, Будапешт, Варшава, Пхеньян, на семинарах КП НТП в г. Братислава, Москва, а также на конференциях:

"**Neumann János**" в г. Секешфехервар 1983, в г. Будапешт 1987.

"Международная конференция КНВВТ по искусственному интеллекту" в г. Пэеньян 1988.

По результатам исследований опубликованы следующие работы

1. Деметрович Я., Ханнак Л., Ремжё Т., Урбански Ф., ЛАТОР-профессиональная система управления с базой данных для локальных сетей. - Международная конференция по искусственному интеллекту, Пхеньян, КНДР, 1988. - с. 225-247.

2. Деметрович Я., Ханнак Л., Ремжё Т., Урбански Ф., Об информационной системе ABLAK. - Международная конференция по искусственному интеллекту, Пхеньян, КНДР, 1988. - с. 247-252.

3. Ремжё Т., Проблемы интеграции в базах данных филиалов страховых учреждениях, MTA SZTAKI Közlemények, 1988. 203/1988. - с. 121-135.

4. Andó Gy., Gyepesi Gy., Hannák L., Remzső T., LATOR - a Database Management system for Local Area Networks. Basic Concepts. MTA SZTAKI Budapest, 1987.

5. Andó Gy., Gyepesi Gy., Hannák L., Remzsö T., LATOR - a Database Management system for Local Area Networks. The Server Program. MTA SZTAKI Budapest, 1987.

6. Andó Gy., Gyepesi Gy., Hannák L., Remzsö T., LATOR - a Database Management system for Local Area Networks. Database Definition. MTA SZTAKI Budapest, 1987.

7. Andó Gy., Gyepesi Gy., Hannák L., Remzsö T., LATOR - a Database Management system for Local Area Networks. The MISS Multitask Interface to DOS 3.1. MTA SZTAKI Budapest, 1987.

8. Andó Gy., Gyepesi Gy., Hannák L., Remzsö T., LATOR - a Database Management system for Local Area Networks. The PASCAL Interface. MTA SZTAKI Budapest, 1987.

9. Andó Gy., Gyepesi Gy., Hannák L., Remzsö T., LATOR - a Database Management system for Local Area Networks. The MICROSOFT-C Interface. MTA SZTAKI Budapest, 1987.

10. Andó Gy., Gyepesi Gy., Hannák L., Remzsö T., LATOR - a Database Management system for Local Area Networks. Utilities. MTA SZTAKI Budapest, 1987.

11. Andó Gy., Gyepesi Gy., Hannák L., Remzsö T., LATOR - a Database Management system for Local Area Networks. The LATROQ Guide. MTA SZTAKI Budapest, 1987.

12. Demetrovics J., Hannák L., Remzsö T., Urbánszki F., LATOR - a Database Management System for Local Area Network, In: Artificial Intelligence III., Methodology, systems applications (Proc. of the Third International Conference on Artificial Intelligence, Varna, 1988.) Ed.: O'Shea T., Sgurev, V., North - Holland, 1988. pp 347-355.

13. Remzsö, T., Computerized Data Base Management Systems, Budapest, Információ/Elektronika, 4 /1979, pp 199-206

14. Remzsö T., Data Base Management Systems (in Hungarian), Technical University of Budapest, Doctoral Theses, 1979., p 165.

15. Remzsö T., Computer Aided Information System in the ELEKTROMODUL, 2th International Conference **Neumann János**, Hungary, Székesfehérvár, 1983.

16. Remzsö T., Industrial Computer Aided Information System Based on the Distributed Data Base Management - A Case Study , MTA SZTAKI Tanulmányok, Budapest, 147/1983, pp 169-174.

17. Remzsö T., Office Automation and data processing. - MTA SZTAKI Tanulmányok, Budapest,194(1986), pp 191-201.

18. Remzsö T., Computer aided management system in the Hernád "Március 15." Agricultural Co-operative (A Case Study). - MTA SZTAKI Tanulmányok, Budapest, 194(1986), pp 183-190.

Автор выражает искренную благодарность своему научному руководителю члену-корреспонденту АН СССР А.А. Стогнию за научное руководство диссертационной работой и постоянное к ней внимание.

Автор признателен сотрудникам Горсистемотехника и ИИВТА ВАН и В.В. Колиньку за внимание к работе.

ГЛАВА 1.
Многоуровневые распределённые системы обработки информации

В выражении "многоуровневая распределённая система обработки информации" скрыто по крайней мере три взаимодополняемые и взаимоуглубляющие понятия:

1. Последовательная обработка информации.
2. Распределённая обработка информации.
3. Многоуровневая распределённая обработка информации.

Для изучения общей проблематики примем метод последовательного анализа.

Понятие "информационные системы" было введено и изучено в работах В.М. Глушкова / 5. /, А.А. Стогния / 30. / ,Е.Л. Ющенко /36., 37. /. Далее системы обработки информации развивались в различных направлениях:

- в направлении создания пользовательских интерфейсов ;
- в направлении создания информационных моделей систем обработки информации - Криницкий Н.А. / 14., 15. /;
- в направлении оптимального использования информационных ресурсов Мартин Дж. / 23., 62., 63. /;
- в направлении создания универсальных подходов в описании информационных систем с использованием теории распознавания Стогний А.А. /29., 30. ,31./, Колинько В.В. / 16.,17.,18.,19. /, Кондратьев А.И. / 29., 31./.

В дальнейшем будем придерживаться трактовки содержательной

обработки информации предложенной в работе А.А. Стогния и А.И Кондратьева /29./, в которой процесс обработки представлен как совокупность следующих этапов (рис.1.1.)

Рис. 1.1.

Этапы обработки информации

В этой схеме существовенно то, что этапы обработки выделены как независимые, что будет использоваться при анализе информационной модели. В / 29. / предложен подход, при котором алгоритмы обработки информации на всех выделенных этапах рассматриваются как алгоритмы распознавания. Это даёт возможность единообразного описания всех этапов обработки. В диссертационной работе не ставится задача глубокого изучения алгоритмов на конструктивном и нормативном уровнях. Алгоритмы используются для удобства построения , изучения информационной модели и выработки технологии и рекомендаций создания конкретных систем распределённой обработки информации.

Введем ряд определений, которые потребуются нам для дальнейщего изложения.

Пусть $S=\{S_1, S_2, \dots, S_m\}$ - множество информационных обьектов, которое может быть разбито на непересекающиеся классы $K_1, K_2, \dots, K_n$.

$$\bigcup_{i=1}^{n} K_i = S.$$

Каждый обьект задается своим информационным описанием

$$S_j = \{\bar{a}_1^j, \bar{a}_2^j, \dots, \bar{a}_k^j\}$$

где a_j - атрибут, $\bar{a}_j$ значение атрибута a_j.

$\bar{a}_j \in M_j$, $j = \overline{1,k}$, M_j - множество значений j-ого атрибута. Содержательно информационное описание S_i - может трактоваться по разному: как объект хранящийся в БД; как объект, который необходимо разыскать в БД или как информационный запрос.

a_j, $j = \overline{1,k}$ можно трактовать как атрибуты, описывающие объекты,

M_i, $i = \overline{1,k}$ как домены атрибутов.

Таким образом $(a_1, a_2, \dots, a_k)$ - отношение, а S_i - имя i-ого экземпляра отношения $S_j = (\bar{a}_1^1, \bar{a}_2^1, \dots, \bar{a}_k^1)$.

В общем случае множество S представляет собой набор реализаций отношения.

Пусть γ, I, $\overline{M}$, C, O обозначают соответственно этапы обработки информации:

γ - предварительная подготовка;

I - ввод информации;

M - хранение;

С - содержательная обработка;

О - вывод информации.

Тогда через $A^{У}$, A^{I}, $A^{\overline{М}}$, $A^{\overline{С}}$, $A^{О}$ обозначим соответствующие алгоритмы обработки.

Введем понятия задачи распознования и задачи поиска.

Под задачей распознавания **Z** (соответствующий алгоритм решения обозначии через A^{Z}) будем понимать задачу отнесения информационного описания объекта $\tilde{S}$ к одному из классов K_j, $j = \overline{1,n}$

$\tilde{S} \in S \vee \tilde{S} \bar{\in} S$.

Под задачей поиска **P** (соответствующий алгоритм решения обозначания через A^{P}) будем понимать задачу нахождения по информационному описанию объекта $\tilde{S}$ ($\tilde{S} \in S \vee \tilde{S} \in S$) объекта $\hat{S} \in S$, такого что $\hat{S} = \tilde{S}$, где знак "=" подразумеват эквивалентность. Содержательный смысл понятия "эквивалентность" определяется видом алгоритма A^{P}.

В // было показано, что A^{P} является частным случаем алгоритма A^{Z}, тогда, когда в классах K_i, $i = \overline{1,n}$ содержится по одному объекту.

Множества алгоритмов $\{A^{Z}\}$, $\{A^{P}\}$ представляют собой алгоритмы содержательной обработки в информационных системах. Таким образом,

$$\{A^{C}\} = \{A^{Z}\} \cup \{A^{P}\}.$$

Под информационно распознающим алгоритмом $\tilde{A}$ будем понимать последовательное применение алгоритмов из множества

$$T = < \{A^Y\}.\{A^I\}.\{A^M\}.\{A^Z\}.\{A^P\}.\{A^O\} >$$

Под схемой информационно распознающего алгоритма **A** будем понимать последовательность применения алгоритмов из множества **T**. Схему алгоритма **A** будем обозначать $\Sigma(A)$.

Так в информационной системе реализующий поиск (обработка информации определяется последовательным выпольнением всех этапов) схема информационно-распознающего алгоритма будет следующей

$$\Sigma(A) = A^Y \cdot A^I \cdot A^M \cdot A^P \cdot A^O ,$$

где "$\cdot$"знак операции последовательного применения алгоритмов.

Информацию, получаемую на каждом этапе, обозначим через I(A).

Тогда в приведеном алгоритме последовательность преобразования информации будет выглядеть так:

$I(A^Y) \rightarrow I(A^I) \rightarrow I(A^M) \rightarrow I(A^P) \rightarrow I(A^O)$, где "$\rightarrow$" обозначает передачу информации.

Если в информационной системе информация, получённая в результате содержательной обработки снова подаётся на вход системы, то схема соответствующего преобразования информации будет такой

$$I(A^{Y}) \rightarrow I(A^{I}) \rightarrow I(A^{M}) \rightarrow I(A^{P}) \rightarrow I(A^{O}).$$ *

Понятно, что схем передачи информации (и соответственно схем информационно-распознающих алгоритмов) может быть множество.

Тогда под информационной моделью **M** мы понимаем следующий кортеж $M=<T, \{\Sigma(A)\}, S>$.

Приведенная модель достаточно хорошо описывает информационные системы функционирующие в рамках одной ЭВМ и выражает то, что мы назвали системой последовательной обработки информации.

Однако в середие $70^{\underline{X}}$ годов начали создаваться системы распредельнной обработки информации.

* Схема информационно распознающего алгоритма почти всегда совпадает со схемой преобразования информации. Поэтому в дальнейшем мы будем пользоваться одним или другим обозначением в зависимости от необходимости.

Одновременно с созданием систем, осуществляющих распределённую обработку информации, появились системы многоуровневой обработки информации, где под уровнями чаще всего понимают уровень интерфейса с пользователем, уровни управления и ответственности:

- уровень администратора БД;
- уровень системного программиста;
- уровень конечного пользователья с органиченной ответственностью;
- уровень привелигированного пользователя и т.д.

Кратко охарактеризуем эти уровни.

Администратор системы выполняет следующие функции:

- создаёт и уничтожает БД;
- отвечает за целостность БД;
- устанавливает уровни пользователей и присваивает им шифры,
- разрешает конфликты;
- ведёт системный журнал и т.д.

Таким образом, администратор БД является лицом (или группой лиц), ответственным за внутренний мир БД.

Системный программист обеспечивает:

- нормальное функционирование системы;
- вносит корректировки;
- исправляет ошибки и т.д.

Конечный пользователь с ограниченной ответственностью получает доступ лишь к определённой части данных, как правило, не имеет права

изменять БД и ведёт диалог, навязанный ему системой.

Привелигированный пользователь имеет возможность изменять БД и сам определяет структуру диалога.

Приведенная классификация не является устоявшейся и не претендует на полноту.

Понятно, что введённая информационная модель не может описать распределённую обработку и требует корректировки. Таким образом, одной из целей, поставленной в диссертационной работе, является расширение этой модели и преломление её в реальной распредельнной системе LATOR.

1.1. Распределённая обработка деловой информации

Любая концепция автоматизированной обработки информации, особенно на современном этапе, практически бессмыслена без продления теоретических исследований в практическую область. Поэтому следующей важной задачей поставленной в диссертационной работе было создание на основе проведенных исследований программных и технических средств, позволяющих высокоэффективно обрабатывать деловую информацию в распределённом режиме.

В начале дадим краткую характеристику этой сферы деятельности.

К системам обработки деловой информации относятся следующие системы:

- системы работающие в области "office automation";
- юридические системы;
- системы страхования и др.

Производительность труда служащих контор, бюро и различных учреждений значительно уступает производительности труда промышленных рабочих. Современные средства автоматизации обработки и обмена информации мало удовлетворяют специфическим потребностям руководителей, а возможности обработки данных, как правило, не обеспечивают ефективную обработку информации, в результате чего производительность труда не достигает требуемого уровня.

Вместе с тем, очевидно, что сегодня в сфере учрежденского труда необходимо обеспечить такую же производительность, как на промышленных предприятиях.

Обследование зарубежных учреждений и контор показало, что 80-95 % рабочего времени служащего уходит на обмен информацией и её обработку, поэтому совершенствование информационных процессов играет решающую роль в снижении затрат на административные нужды и позволяет руководящим работникам значительно повысить оперативность и обоснованность принимаемых решений. Оказалось, что администраторам скорее нужны более производительные и эффектувные средства обмена информацией, чем более производительные и мощные ЭВМ.

При автоматизации учрежденского труда и создании электронной конторы автоматизируют такие функции:

- обработку текста;
- функцию электронной почты;
- хранение документов;
- передачу и обработку факсимильной информации;
- дистанционные телевизионные совещания (телесовещания);
- запоминание и обработку голосовой информации;
- персональную обработку документов с помощью ЭВМ;
- воспроизведение и размножение документов;
- обмен информацией между БД;
- контроль испольнения входящей и выходящей корреспонденции;
- обмен данными между потребителями и интеграцию программных средств.

Особенностью современного этапа автоматизации работы служащих является широкое применение персональных ЭВМ, объединяемых в локальные сети, перенос всего документооборота и информационной базы на машинные носители, организация электронного хранения и воспроизведения информации.

В учреждениях всегда приходится иметь дело со смешанными видами информации. Например, в любой отчёт всегда входят числовые данные, пояснительный текст и графические иллюстрации. В настоящее время при помощи ЭВМ можно отдельно работать с числовой и текстовой информацией, а также информацией, представленной в виде простейших таблиц и диаграмм. В то же время, пользователю пока широко не доступна визуальная (графическая) информация, необходимая практически всем пользователям.

При создании новых систем программного обеспечения ставится задача перехода от традиционной (на сегодняшний день) раздельной обработки чисел, текстов, таблиц, графиков к комплексной (интегрированной обработке) и обеспечению обмена деловой информацией на основе формирования цветных графических образов произвольного вида.

Так как автоматизированные учреждения, конторы и бюро должны обрабатывать и передавать большое число электронных документов, содержащих как числовые, текстовые, табличные, так и графические данные, то международная организация по стандартизации приступила к разработке стандартов на электронные документы, содержащие все виды данных.

Следует отметить, что передача документов в электронной системе является более сложной задачей, чем взаимодействие человека при помощи сообщений, поскольку в первом случае требуется организовать взаимодействие между машинами, а необходимые для этого протоколы достаточно сложны и требуют стандартизации.

Как правило, технические средства автоматизации учреждений и контор построены по магистрально-сетевому принципу: у каждого сотрудника на рабочем месте устанавливается рабочая станция, представляющая собой, как правило, либо персональную ЭВМ, объединенную с другими персональными ЭВМ через сетевую магистраль, либо терминал, подсоединённый к мини-ЭВМ, работающей в режиме разделения времени. Из этих двух вариантов рабочей станции с каждым годом все большее предпочтение отдаётся персональным ЭВМ, объединённым в локальную сеть. Однако многие организации пока ещё предпочитают многотерминальную мини-ЭВМ в режиме разделения времени, особенно, если необходимо работать с большими централизованными фондами информации или с очень большими

программами.

Юридические системы и системы страхования включают в себя все перечисленные функции автоматизации учреждений, только в зависимости от специфики, упор делается на те или другие аспекты. Так в юридических системах предпочтение отдаётся точности и коректности получаемых справок, а как результат, необходимо более детальное продумывание организации БД и наличие более содержательного язык запросов.

Системы страхования отдают предпочтение точности и функциональной полноте операций производимых над данным.

Операции в системе страхования должны отвечать следующим правилам:

1. Учитывать как международные соглашения (например, для страховки автомобилей) так и национальные особенности страхования.
2. Распределённая система страхования должна единообразно работать во всех пунктах страховании (Обычно их бывает достаточно много. Система **LATOR** поддерживает около 640 таких пунктов.)
3. Работа в системе должна поддерживаться как в диалоговом, так и в пакетном режимах.

В диалоговом режиме осуществляется обновление БД (при проведении страховых операций), непостредственное страхование (таксация), выдача компенсацией и др. В пакетном режиме производится автоматическое страхование , изготовление распечаток, периодическая архивизация данных.

1.1.1. Технические средства.

Создание систем многоуровневой распределённой обработки информации требует совершенной технической базы. Как уже упоминалось, в настоящее время такой базой является семейство персональных ЭВМ, которое менее чем за десять лет завоевало себе абсолютное первенство в области обработки деловой информации.

Вычислительные возможности

Семейство микропроцессоров **INTEL** (**I8080** - 8 разрядные, **I8086, I8087 - I80286, I80287-** 16 разрядные, **I80386** - 32 разрядные) обеспечило рост производительности персональных ЭВМ от машин малой производительности: **IBM PC - IBM PC-XT**, через машины средней производительности: **IBM PC-AT, IBM PC-XT TURBO**, до машины высокой производительности **PS/2-Model 80**.

Объём памяти

Одновременно с наращиванием производительности шло увеличение объёмов памяти: ОЗУ увеличилочь с 64 Кбайт до 4-8 МГбайт; накопители на жёстких дисках типа Winshester увеличились от объёма в 10 МГбайт до 300-1000 МГбайт, причём диски до 115 МГбайт монтируются непосредственно в корпусе ЭВМ.

Возможности дисплеев

Увеличение возможностей дисплея, определяется прежде всего увеличением возможностей соответствующих адаптеров. Самые распространённым адаптером для монохронных дисплеев является **HERCULES** , а для цветных **CGA (Color Graphics Adapter)**. Однако с развитием систем отображения (широко применяемых сейчас а в системах обработки деловой информации) были созданы более мощные по цветовым, визуальным и манипуляционным возможностям адаптеры **EGA (Enhanced Graphics Adapter), PGA (Professional Graphics Adapter)** и др.. Таким образом, в настоящее время персональные ЭВМ обеспечены широкими визуальными возможностями, предоставляющие разработчикам систем право искать оптимальное отношение между выразительными способностями технических средств и возможностями восприятия информации пользователями.

Манипуляторы

Развитие персональных ЭВМ вызвало бум развития манипуляторов, позволяющих на экране дисплея адресовать положение курсора, используя неклавишные методы. Особое распространение получили два типа манипулятора:

joystick- имеющий вид рукоятки с четырьмя степенями свободы и использущийся в тренажерах и играх;

mouse(мышь)- имеющий вид полусферического устройства, которое пользователь перемещает по гладной поверхности.

Использование этих манипуляторов значительно повысило комфортность работы с системами, особенно с меню управляемыми системами, где производительность (за счёт манипуляторов)

повышается в несколько раз.

Нельзя не отметить ещё один вид специализированного адресования области на экране дисплея - сенсорное адресование. Такие системы удобны при работе с неквалифицированным пользователем, когда вся информация для выбора представлена на экране.

Возможности печати

Удобство использования систем обработки деловой информации в немалой степени определяется возможностями получения качественной твёрдой копии документа (копирование, редактирование с последующей печатью, подготовка выходных документов и др).

Поэтому были предприняты специальные усилия для создания печатающих устройств, работающих в самых разнообразных режимах, с печатью по качеству приближающемся к полиграфическому. Так ЭВМ серии **Macintosh** (фирма **Apple**) снабжаются высококачественной печатью и часто используются для подготовки документов.

В настоящее время получили распространение так называемые настольные типографии (desktop publishing systems), которые используются разными организациями для тиражирования своей продукции.

Также имеются высокоточные лазерные печатающие устройства (как одно так и многоцветные), которые применяются как выходные печатающие терминалы.

Для документирования изображений (деловая графика, схемы, рисунки) используются одно и многоцветные принтеры (чаще всего построенные на струйном или лазерном принципе отображения), а также разообразные графопостроители.

Сетевые возможности

95 % используемых персональных ЭВМ объединены в локальные сети. Количество выпускаемых сетей уже перевалило 100. К наиболее распостраненным можно отнести: **PC-NET, CORVUS OMNINET, NOVELL, ETHERNET, 3COM, 10NET, DLINK** , и др. Теннология использования этих сетей самая разнообразная: от простой витой пары с последовательной передачей эстафеты до сложных соединений типа звезда, сетей с коммутированием запросов специальными дорогостоящими транспортерами.

Однако, как будет показано в диссертационной работе, наличие технических сетевых возможностей требует ещё хорошего системного и прикладного математического обеспечения, создание которого в общем случае является нетривиальной задачей.

Возможности ввода инфирмации

Быстрота и качество ввода во многом определяет качество работы с системой. Так если говорить об информационных системах использующих БД о населении или библиотечных системах, то трудоёмкость первоначального заполнения БД может достигать сотен человеко лет. А если говорить о графических БД, то ввод изображения вручную в ЭВМ часто просто невозможен.

Сейчас разработаны устройства для быстрого ввода информации в персональные ЭВМ. Правда следует отметить, что они достаточно дорогие:

Сканеры - устройства, позволяющие вводить растровое изображение в цветном или черно-белом формате.

Диджитайзеры - устройства, позволяющие вводить контурные изображения.

Читающие автоматы - вводят и распознают различные шрифты.

1.1.2. Математическое обеспечение

Самые совершенные технические устройства ничего сами по себе не решают. Они требуют подчас колосальных затрат на написание программ.

Бум на рынке технических средств персональных ЭВМ вызвал бум на рынке производителей математического обеспечения. В короткие сроки (3-5 лет) было создано огромное количество программных продуктов. Так только к персональным ЭВМ фирмы **IBM** продаётся свыше 20.000 различных пакетов. Кратко охарактеризуем различные классы программных средств персональных ЭВМ.

Системное математическое обеспечение

Операционные системы

Стандартом "de-facto" в операционных системах (**ОС**) стала **MS-DOS** и её дальнейшее развитие **OS2**, применяемые для **IBM** совместимых машин. Правда, сейчас получают распростанение **UNIX** подобные системы: **XENIX**, **ONIX**, и др. В целом существует порядка 10 достаточно распространённых **OS** на персональных ЭВМ.

Языки и компиляторы

Трудно назвать язык, который не был бы представлен в спектре математического обеспечения персональных ЭВМ: **BASIC, Turbo-Basic,**

PASCAL, Turbo-Pascal, C, Turbo-C, LOGO, PROLOG, MPROLOG, PROFESSIONAL PROLOG, FORTRAN, PL/I, MODULA-II, и т.д.

Утилиты, отладчики

Разработаны очень удобные утилиты (форматирование, копирование- **PCTOOLS, NORTON**, и т.д.) и отладчики (**Code View, Periscop**), использующие возможности манипуляторов и цветного дисплея. Можно с уверенностью сказать, что по комфортности использования этот класс математического обеспечения на персональных ЭВМ превосходит представителей этого класса на других типах ЭВМ.

СУБД

Существует более 50 промышленно-используемых СУБД в основном реляционного типа. К наиболее распостраненным следует отнести **dBASEIII PLUS, dBASE IV, ORACLE, Data Ease**. Они имеют все черты современных СУБД и практически не отличаются от них по мощности.

Текстовые редакторы

Персональные ЭВМ завоевали популярности прежде всего из-за их возможностей редактирования произвольного текста и документов. Такие редакторы как **MS-WORD, WORDSTAR, LEXICON** значительно упрощают жизнь как пользователям систем обработки информации, так и профессиональным программистам, готовящих и отлаживающих программы.

Графические редакторы

Этот класс математического обеспечения получил распространение в связи с возможностью растровых дисплейев. Такие редакторы как **Graphics Partner, PC-Paintbrush, Graftalk, Fastgraphs** и др. позволяют пользователю нарисовать, закросить, изменить, выбрать и т.д. графические примитивы. Более развитые редакторы используют графические меню, позволяющие выбрать стиль рисования, совместить надписи и изображения и т.д.

Spreadsheet

Или крупноформатные электронные таблицы исторически были первыми программами использующимися в конторах для бухгалтерской деятельности, заполнения ведомостей материально-технического снабжения и др.

Интегрированные средства

Этот класс математического обеспечения объединяет разные возможности : СУБД, текстовых реадкторов, spreadsheet и др. Такие пакеты позволяют обрабатывать информацию в одном стиле, используя общие БД. К этим средством можно отнести пакеты: Knowledgeman, Symphony, LOTUS 1-2-3.

Сетевые средства:VIDEOTEX,TELETEX

Объединение персональных ЭВМ в локальные сети послужило толчком для разработки специфических программ, позволяющих распределённую обработку информации. Так, например, системы VIDEOTEX имеют возможность информировать покупателей магазинов о наличии товара с представлением внешнего вида товаров. Такие справочные системы позволяют не только передавать по сети алфавитно-цифровую, но и графическую информацию.

1.2. Интеграция систем обработки данных

Мы уже упоминали об интегрированных системах обработки информации. Сейчас необходимо исследовать интеграцию систем обработки данных в целом, поскольку из понимания процесса интеграции можно делать вывод о преимуществах или недостатках той или иной системы.

Интеграцию мы будем рассматривать в следующих аспектах:

1. Интеграция по функциям обработки внутри одной персональной * ЭВМ.
2. Интеграция БД (в том числе и неоднородных) внутри одной персональной ЭВМ.
3. Интеграция по функциям обработки и интеграция различных БД внутри одной персональных ЭВМ.
4. Интеграция по функциям обработки в рамках локальных сетей.
5. Интеграция БД в рамках локальных сетей.
6. Интеграция по функциям обработки и интеграция БД в рамках локальных сетей.

* В зависимости от контекста мы будем пользоваться равноначными тарминами:персональная ЭВМ, ЭВМ, машина.

1. Интеграция по функциям наиболее законченное оформление получила в интегрированных системах. В таких системах, как правило, выбирается центральная функция, она становится ведущий, а все остальные подчинены ей по форме представления данных и идеологии обработки.

Так в пакете Knowledmen в качестве такой функции выбрана СУБД. Все остальные : spreadsheet, текстовый реадктор имеют явно выражению тенденцию обеспечивать функционирование СУБД, хотя могут работать и независимо.

В пакете SYMPHONY в качестве основной компоненты выбрана бесконечная электронная плоскость (spreadsheet) .

В пакете FRAMEWORK основным элементом обработки является фрейм (в трактовке М. Минского /22. /). Все объекты называются фреймами.

Интеграция по функциям внутри одной персональной ЭВМ достаточно прозрачна, поскольку все ресурсы машины локализованы, централизованы и выполняются одним процессором под управлением одной операционной системы. Совсем другое дело, когда обработка разнесена по ЭВМ, включенных в локальную сеть.

Представим себе, что обработке подвергается документ в некотором учреждении. Причём на одной ЭВМ документ печатается (секретарь- функции текстового радактора), на другой проверяется, заполняется и передаётся на хранение (служащий - функции spreadsheet и СУБД) , а на третей из ряда хранащихся документов получается сводная характеристика в виде твёрдой копии со столбчатыми диаграмами (управляющий-функции СУБД, графического и текстового редакторов.) Сразу возникают возможности колизий по потверждению готовности, разрешению конфликтов по неподготовке документов и другое.

Ситуация как в 1ом так и в 3 ем случае усложняется, если в интеграцию вовлечены разные (в том же числе и неоднородные) БД.

2. Пусть на одной ЭВМ хранятся БД о разных сторонах деятельности учреждения : БД1- служащие, БД2 - БД по материально-техническому снабжению, БД3 - БД ведомостей заработной платы. Пока в каждой БД обрабатывается по одному запросу и одним каким-то действием (например справка) то всё относительно благополучно (рис. 1.2.)

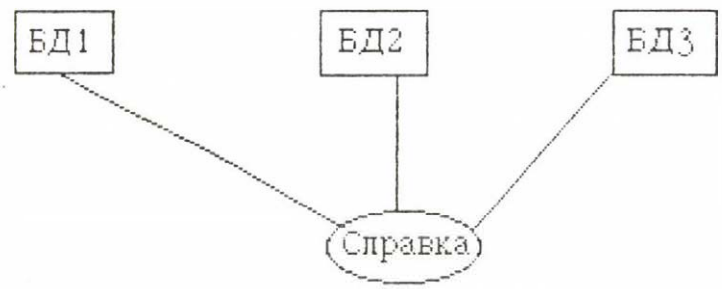


Рис.1.2. Интеграция по данным внутри одной персональной ЭВМ

3. Если к этому подключить возможность интеграции по функциям обработки разных БД, то картина получается более сложной. Так, пусть нам необходимо взять список служащих, проверить кому из них выдавались магнитные носители в прошлом месяце, для этих служащих составить ведомость заработной платы, заполнить её, поместить в БД3 и распечатать в виде диаграммы (рис.1.3.)

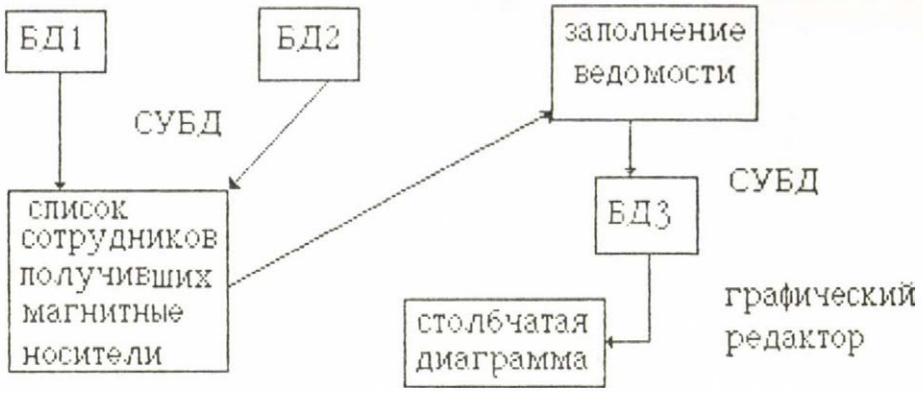


Рис. 1.3. Интеграция по функциям обработки и интеграции БД внутри одной ЭВМ

4.-5. Если БД рассредоточены на разных ЭВМ и интегрированны по функциям обработки, то функционирование такой локльной сети потребует большой не только программно-технической, но и организационно-методологической деятельности. В противном случае такая система может постоянно рождать неразрешимые конфликты между программами, запросами, техническими средствами и т.д.

В настоящее время ведутся интенсивные работы по интеграции неоднородных баз данных. Эти вопросы освещены в трудах Л.А. Калиниченко /12/ и вошли в программы международного сотрудничества.

Системы интеграции неоднородных БД

Системы интеграции неоднородных БД и баз знаний представляют собой распределённые, мобильные аппатратно- программные системы управления непроцедурным решением задач представления знаний в сфере разнотипных ЭВМ, традиционных и нетрадиционных средств

программирования и программ, написанных на различных языках, (в сфере разнотипных СУБД и неоднородных БД и баз знаний).

Системи интеграции ориентированы на применение в качестве инструментальных средств создания систем обработки знаний и данных (экспертных систем, систем принятия решений, и т.д.) в разнообразных областях человеческой деятельности при использовании отдельных ЭВМ или неоднородных информационно - вычислительных систем (НИВС). Система интеграции должна обеспечивать возможность использования существующих (в общем случае неоднородных) и программ, написанных на различных языках программирования.

Языковые средства системы интеграции должны предоставить возможность структуированного, модульного описания систем обработки знаний при использовании разнообразных входящих в состав НИВС систем программированя и СУБД (в частности, в рамках работ по созданию машин новых поколений.) Средства программирования должны обеспечивать эффективное управление решением задач при использовании как традиционных ЭВМ, так и спецпроцессоров (таких как машины логического вывода), используемых в разнообразных сочетаниях, включая многопроцессорные и распределённые системы.

1.3. Задача создания информационной системы страхования

Произведенный анализ систем, средств и моделей обработки информации позволяет сделать ряд выводов, которые в конечном счёте и определили предмет и задачи диссертационного исследования.

1. Существующие информационные модели недостаточны для описания распределённой, многоуровневой обработки информации

и требуют уточнения. Причём уточнение требуется не только и не столько для изучения алгоритмов обработки, а для технологически обоснованного создания таких систем.

2. В качестве технических средств создания информационных распределённых систем целесообразно выбирать класс персональных ЭВМ, как наиболее отвечающих задачам создания высокоэффективных систем обработки.
3. Хотя существует сетевое обеспечение локальных систем обработки информации, для развития национальных информационных систем, имеющих внутреннюю специфику, есть смысл создавать собственные системы. Это обеспечивает технологическую независимость и конкурентноспособность как на внутреннем, так и на внешнем рынках.
4. Система страхования принадлежит к классу систем, отвечающих всем признаком распределённых многоуровневых систем обработки информации.

ГЛАВА 2.

Информационная модель распределённых систем обработки данных

Настоящая глава посвящена расширению информационной модели обработки информации $M = \langle T, \{\tilde{E}(\tilde{A})\}, S \rangle$ для случая распределенной обработки и построению на её основе информационной модели систем страхования разных уровней, которые используются при создании системы страхования в ВНР ABLAK.

2.1. Различные формы распределённой обработки.

В первой главе были перечислены различные формы распределённой обработки внутри одной персональной ЭВМ и в рамках локальной сети. Теперь следует выяснить соотношение между "распределением" и "интеграцией". Эти понятия являются взаимодополняющими. Действительно, распределение функций обработки может подразумевать их интеграцию внутри системы обработки, а распределение БД на разных ЭВМ часто подразумевает интеграцию их обработки внутри локальной сети. Поэтому эти термины в дальнейшем противопоставляться не будут, а их применение будет определяться необходимостью.

2.1.1. Интеграция по функциям обработки внутри одной ЭВМ

Пусть $R=\{R_1, R_2, \ldots, R_n\}$ множество функций, которые реализуют содержательную обработку информации над множеством информационных объектов $S=\{S_1, S_2, \ldots, S_k\}$.

Так R_i , $1< i < n$ может быть функцией текстового редактора, функцией **spreadsheet**, и т.д. Понятно, что эти функции выполняются над некоторыми конкретными объектами из S (одним или несколько), в результате выполнения которых мы получаем новый объект (или несколько объектов) $\hat{S} \in S$ $R_j(S_j) = \hat{S}$, $i=\overline{1,n}$, $j=\overline{1,k}$.. Объект S_j должен быть предварительно выбран из БД, в результате решения задачи поиска и (или) распознавания. Причём в зависимости от качества алгоритма объект S_j может быть выбран правильно или неправильно . Изучим этот вопрос.

Алгоритм распознавания A^Z

Результат работы алгоритма A^Z над объектом $\tilde{S}$ может быть следующим:

1. Алгоритм отнёс объект $\tilde{S}$ в класс K_i.

2. Алгоритм не отнёс объект $\tilde{S}$ ни в один из классов.

3. Алгоритм отказался от классификации $\tilde{S}$.

Тогда результат работы алгоритма A_i по i-ому классу будет выражаться следующим соотношением:

$$A_i^Z(\tilde{S}) = \begin{cases} 1, \text{если} \quad \tilde{S} \in K_i; \\ 0, \text{если} \quad \tilde{S} \overline{\in} K_i; \\ \Delta, \text{если алгоритм отказался от классификации} \end{cases}$$

При этом возможны ошибки двух видов:

Алгоритм A^Z отнёс $\tilde{S}$ в K_i, но $\tilde{S} \overline{\in} K_i$.

Алгоритм A^Z не отнёс $\tilde{S}$ в K_i, но $\tilde{S} \in K_i$.

Если $\tilde{S}$ запрос в информационной системе, то качество работы алгоритма распознавания играет важную роль в эффективности системы обработки.

Содержательная интерпретация обработки объекта $\tilde{S}$ в системах обработки информации может быть следующая.

1. Пользователю необходимо выбрать объект , являющийся по некоторым критериям подобным классу объектов в БД и, если такой существует, то передать его графическому редактору. Подобная задача может иметь место в криминалистике, когда необходимо составить фоторобот объекта, который совпадал бы с фотороботами, хранящихся в БД с точностью до класса и распечатать его на принтере.

2. Необходимо описать предмет страховки с точностью до класса и, если такой имеется, поместить его в БД в наиболее подхоящий подкласс.

Алгоритмы поиска A^P

Результат работы алгоритма поиска объекта $\tilde{S}$ в БД по некоторому отношению эквивалентности "=" может быть следующим:

1. Алгоритм нашёл в БД объект $S_i = \tilde{S}$, $S_i \in S$

2. Алгоритм не нашёл в БД ни одого объекта удовлетворяющему правилу эквивалентности..

3. Алгоритм не смог определить эквивалентность (например, вследствие недостаточности информации.)

Тогда результат работы алгоритма на объекте $S_i \in S$ будет выражаться следующим соотношением:

$$A_i^P(\tilde{S}) = \begin{cases} 1, \text{если } \tilde{S} = S_i; \\ 0, \text{если } \tilde{S} \neq S_i; \\ \triangle \text{ если алгоритм отказался от установления эквивалентности.} \end{cases}$$

При этом, как и в случае алгоритма распознавания возможны ошибки двух видов:

Алгоритм A^P определил, что $\tilde{S} = S_i$, но $\tilde{S} \neq S_i$.

Алгоритм A^P определил, что $\tilde{S} \neq S_i$, но $\tilde{S} = S_i$.

В любой информационной системе алгоритмы поиска играют ведующую роль, поскольку определяют дисциплину доступа к информации.

Содержательно введенный алгоритм поиска может быть проинтерпретирован следующим образом.

1. Из БД необходимо выбрать объект для редактирования по некоторому критерию эквивалентности (правилу доступа). Если такой объект в БД имеется , то после редактирования заменить его на новый.

2. Применительно к системам страхования это может быть поиск соответствующей страховки для внесекия изменений в адрес клиента и (или) условий страхования.

Таким образом, с некоторой степенью достоверности объект может быть получен для обработки функцией $R_i \in R$.

Дополним информационную модель M множеством функций обработки. Тогда информационная модель M_1, включающая интеграцию по функциям обработки внутри одной ЭВМ будет следующий:

$$M_1 = < T, \{ E (\tilde{A}) \}, S, R >$$

Поскольку внутри одной ЭВМ в каждый момент времени t может выполняться только одна функция, то схемы информационно-распознающих алгоритмов будут линейны и не требовать синхронизаций по времени.

Покажем это на примере нескольких задач.

Задача страхования машины от угона

Содержательная последовательность действий при страховании может быть следующей.

1. Реквизиты клиента (персональный идентификатор, фамилия, имя, адрес жительства, марка машины, сумма страховки и т.д.) вносятся в БД.

2. В БД проверяются внесённые данные на соответствие классу страховок от угона.

3. Если система не установила нарушений, то данные заносятся в БД клиентов и выдаётся соответствующий документ.

Проиллюстрируем работу системы соответствующим информационно-распознающим алгоритмом, работающим в рамках модели M_1.

Пусть $S=\{ S_1, S_2, ..., S_k \}$ - БД клиентов;

$S_1 = \{ S_1^1, S_2^1, ..., S_{k1}^1 \}$ - база данных условий страхования;

$\tilde{S} = (a_1, a_2, ..., a_n)$ - реквизиты клиента:

a_1 - персональный идентификатор

a_2 - фамилия, и т.д.

$A^I (\tilde{S})$ - алгоритм ввода в БД;

$A^2 (\tilde{S})$ - алгоритм распознавания вычисляющий:

$$A_i^Z(\tilde{S}) = \begin{cases} 1, \text{если } \tilde{S} \text{ соответствует классу } K_i; \\ 0, \text{если } \tilde{S} \text{ не соответствует классу } K_i; \\ \triangle, \text{если классификация невозможна.} \end{cases}$$

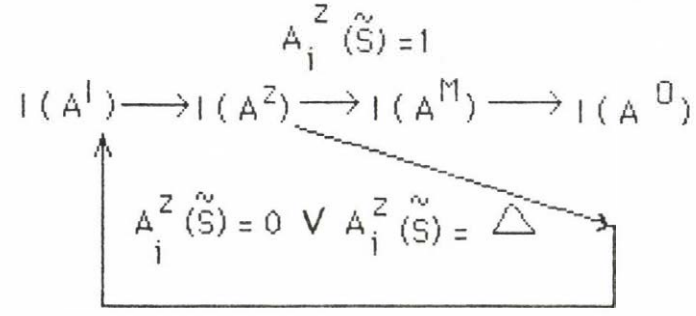
$\bigcup_i K_i = S_1$ (условия страховки).

Пусть в данном случае K_i - условие страховки от угона автомобиля.

$A^M(\tilde{S})$ - алгоритм помещения $\tilde{S}$ в S, т.е. $S' = S \cup \tilde{S}$

$A^O(\tilde{S})$ - алгоритм выдачи документа о страховании.

Тогда последовательность преобразования информации будет следующей



* (2.1.)

где $A_i^Z(S) = 1$ - условие выполнения передачи.

* Чтобы не затуманивать понимание мы не будем указывать в схемах такие тривиальные действия как останов, запуск, и т.д.

$A_i^Z(\tilde{S}) = 0 \vee A_i^Z(\tilde{S}) = \Delta$ обозначает, что в случае, когда алгоритм $A^Z(\tilde{S})$ определил, что данные по клиенту не соответствуют условию страхования по угону автомашины или если такая классификация произведена быть не может (например, когда клиент не сообщил достаточно данных), тогда диалог системы снова возвращается на ввод информации для уточнения.

Пусть через некоторое время клиент явился в страховую контору с целью изменить условия страхования (увеличить сумму страховки) и откорректировать свои реквизиты (изменилось место жительства). Тогда последовательность действий будет следующая:

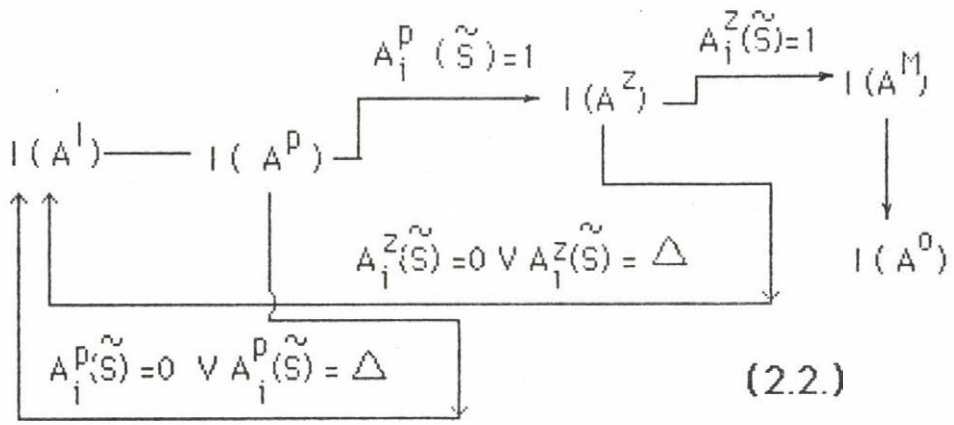
1. Вводятся реквизиты клиента для идентификации.
2. Система производит поиск соответствующей страховки.
3. Вносится корректировка.
4. В БД снова проверяется соответствие новой страховки классу страховок от угона.
5. Если система установила соответствие, то новая страховка заносятся в БД клиентов, старая изымается и выдаётся новый документ страхования.

Пусть $\tilde{S}$ - новые реквизиты клиента.

$A^P(\tilde{S})$ - алгоритм поиска вычисляющий:

$$A_i^p(\tilde{S}) = \begin{cases} 1, \text{если } S = S_i,\ S_i \in S^*; \\ 0, \text{если } S \neq S_i; \\ \triangle, \text{установление эквивалентности невозможно.} \end{cases}$$

Последовательность преобразования информации для решения данной задачи будет следующей:



(2.2.)

*** Отметим, что условие может выполняться $\forall i : (A_i^p (\tilde{S}) = 1)$,**

т.е. перебор идёт по всем $S_i \in S$.

Полная схема преобразования информации в информационной системе для решения задачи первоначальной регистрации клиента и последующего изменения будет заключаться в последовательной композиции схем преобразования (2.1.) и (2.2.).

Необходимо различать внутренние функции информационной системы, выражающиеся алгоритмами из множества **T** и функции пакетов прикладных программ из множества **R** . Функции $R_i \in R$ строятся на основе функций **T** , но не исчерпываются ими. Так в любом пакете построеном над СУБД (на основе системы программирования этой СУБД) есть функции ввода, запоминания, поиска, распознавания, но они скрыты внутри пакета, выполняющего более специальные функции: редактирование, отображение, и т.д.

Более того, в инструментальных средствах, как правило, "в чистом виде" не существуют ни функции множества **T** ни **R** , а пользователю предоставляется некоторый язык программирования, на котором и пишется то или инное приложение. Так в СУБД dBASE III применяется собственный язык программирования высокого уровня, в системе **LATOR** - языки **Turbo PASCAL** и **Microsoft C** дополненные некоторыми конструкциями. Но для изучения информационных моделей удобно пользоваться делением на функции множеств **T** и **R** .

При рассмотренной модели M_1 (в проанализированном примере) по существу были использованы две БД и несколько функций обработки, реализующиеся в рамках одной ЭВМ.

S - база данных о клиентах;

S_1 - база данных об условиях страховки;

A^I - функция текстового редактора;

A^P, A^z, A^M - функции СУБД.

Однако, для изучения информационных моделей описывающих интеграцию более удобно их отдельное рассмотрение.

2.1.2. Интеграция БД (в том числе и неоднородных) внутри одной ЭВМ

Давайте зададимся вопросом: к чему приводит интеграция неоднородных БД внутри одной персональной ЭВМ с точки зрения алгоритмов обработки и информационной модели.

До настоящего момента мы не рассматривали внутренний вид данных и алгоритмов обработки из множеств T и R.

Мы указали при определении множеств значений атрибутов M_i, $i=\overline{1,n}$, что эти множества обладают некоторой метрикой. Значит мы предполагаем, что на элементах множества M_i существует (или может быть введена) метрика или, по крайнер мере, какое то отношение порядка (хотя бы порядок следования) .

Пусть в ЭВМ одновременно имеются разнотипные БД: БД описания людей - S, БД фотороботов - S_1 и БД их голосов - S_2.

$S_i = (a_i(i), a_2(i), \dots , a_n(i))$.

$S_i^1 = (a_1^1(i), a_2^1(i), \dots , a_{n1}^1(i))$.

$S_i^2 = (a_1^2(i), a_2^2(i), \dots , a_{n2}^2(i))$; $S_i \in S$, $S_i^1 \in S_1$, $S_i^2 \in S_2$.

Тогда на некоторых множествах значений атрибитов может быть введено **обычное эвклидов расстояние** (например, если a_j - рост), на другои некоторой **лексикографическиий порядок** (например , если a_i -имя), а на третьих только **порядок следования** (например, если a_i^1 - цвет волос.) Понятно, что в этом случае алгоритмы обработки будут разными (в большей степени это касается алгоритмов содержательной обработки.) Поясним это на примере алгоритмов поиска.

В алгоритмах класса A^P присуствует понятие эквивалентности двух объектов S_i и $\tilde{S}$, такое что $A_i^P(\tilde{S}) = 1$, если $S_i = \tilde{S}$. В случае, если множество значений атрибутов $(a_1, a_2, \dots, a_n)$ метрическое, то под эквивалентностью будет пониматься обычные правила сравнения. Например, два обьекта S_i и $\tilde{S}$ "эквиваленты", если значения их атрибутов различаются на ε. Тогда алгоритм A_i^P примет вид

$$A_i^P(\tilde{S}) = \begin{cases} 1, \text{ если } \forall i : |\overline{a_i} - \overline{\tilde{a}_i}| < \varepsilon \\ 0, \text{ если } \exists i : |\overline{a_i} - \overline{\tilde{a}_i}| \geq \varepsilon. \end{cases}$$

Кстати, здесь можно проиллюстрировать случай, когда результатом алгоритма поиска будет неопределённость. Пусть условия эквивалентности формулируются следующим образом:

два обьекта S_i и $\tilde{S}$ "еквиваленты", если $\forall i : (\overline{a_i} - \overline{a'_i}) > \varepsilon$

и "неэквиваленты", если $\forall i : (\overline{a_i} - \overline{a'_i}) < \varepsilon$.

Тогда:

$$A_i^P(\tilde{S}) = \begin{cases} 1, \text{ если } \forall i : (\overline{a_i} - \overline{a'_i}) > \varepsilon. \\ 0, \text{ если } \forall i : (\overline{a_i} - \overline{a'_i}) < \varepsilon. \\ \Delta, \text{ если } \exists i : (\overline{a_i} - \overline{a'_i}) = \varepsilon. \end{cases}$$

В случае, если множество значений атрибутов не метрическое (например, набор символов), то под эквивалентностью будем понимать сравнения двух символьных строк в смысле совпадения. Тогда $a_i = a'_i$, если посимвольно совпадают все его значения.

Например, дба обьекта S_i и $\tilde{S}$ "еквивалентны", если посимвольно совпадают значения всех атрибутов.

$$A_i^P(\tilde{S}) = \begin{cases} 1, \text{ если } \forall i : (\overline{a_i} = \overline{a'_i}); \\ 0, \text{ если } \exists i : (\overline{a_i} \neq \overline{a'_i}). \end{cases}$$

Аналогичное рассмотрение может быть произведено и для алгоритмов распознавания. Тогда, составляющие алгоритма A^Z - распознающий оператор и решающие правило будут выбираться с учётом

метрики.

Естественно, возможен смешанный случай, когда в одном обьекте есть атрибуты разной природы. В этом случае алгоритмы будут приобретать вид последовательного выполнения различных алгоритмов.

Рассмотрим пример функционирования такой информационной системы с разнородными БД (на примере криминалистической информационной системы).

Пусть имеется две разнородных БД:

S - БД описание людей;

S_1 - БД фотографий и описаний соответствующих фотороботов.

Пусть было совершено ограбление и есть свидетель видевший грабителя в лицо. Также имеются некоторые данные, позволяющие идентифицировать грабителя (например, размер обуви, рост, и т.д.) Последовательность работы такой системы может быть следующий.

1. По имеющиися атрибутам: размер обува a_i, рост a_j в БД S производится поиск тех обьектов которые совпали по значениям атрибутов a_i и a_j.

$$A_k^P(\tilde{S}) = \begin{cases} 1, \text{ если } & (\bar{a}_i^k = \bar{\tilde{a}}_i^k) \wedge (\bar{a}_j^k = \bar{\tilde{a}}_j^k) \\ 0, \text{ если } & (\bar{a}_i^k \neq \bar{\tilde{a}}_i^k) \vee (\bar{a}_j^k \neq \bar{\tilde{a}}_j^k). \end{cases}$$

Пусть S' множество обьектов, для которых $A_k^P(\tilde{S}) = 1$, $S' \in S$.

2. Производится опрос свидетеля и состовлять описание фоторобота $\tilde{S}'$.

3. Решается задача поиска для обьекта $\tilde{S}'$ в БД S_1. Правило поиска такое: два фоторобота считаются эквивалентным, если совпало (в смысле совпадения значений) значений не менее 50 % атрибутов:

$$A_k^{P1}(\tilde{S}') = \begin{cases} 1, \text{ если } (\Sigma i : (\bar{a}_i^{k,1} = \bar{\tilde{a}}_i^{1})) \geq 0.5\, n_1; \\ \\ 0, \text{ в противном случае.} \end{cases}$$

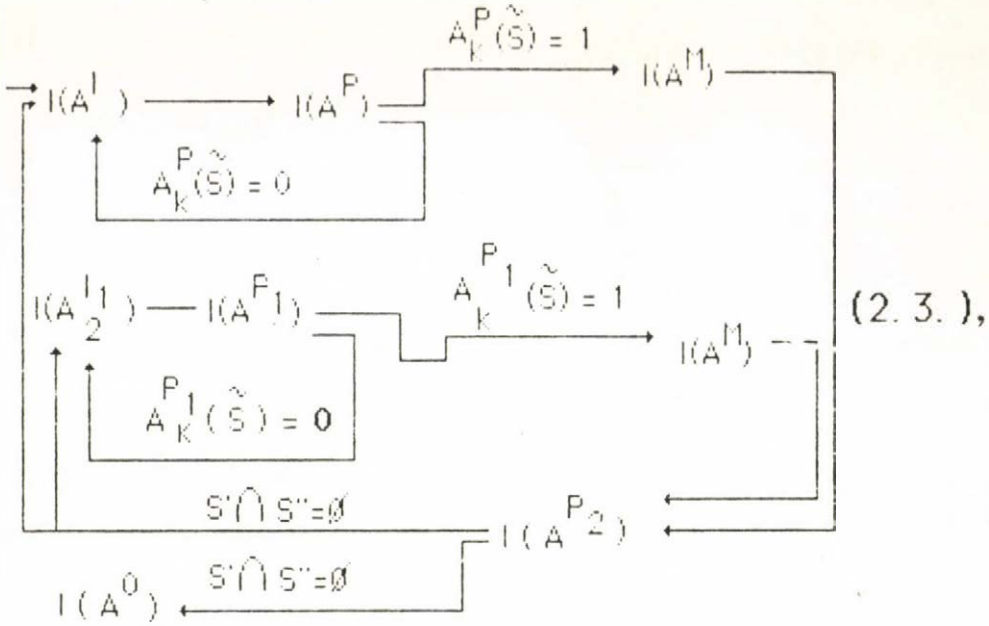
здесь $\bar{a}_i^{k,1}$ - значение i-ого атрибут к-го обьекта множества S_1.

$$S_1 = (a_1^1, a_2^1, \dots, a_{n1}^1)$$

Пусть S'' множество обьектов, для которых $A^{P1}(\tilde{S}') = 1$, $S'' \in S_1$.

4. Определяем пересечение множеств S' и S''. Если $S' \cap S'' \neq 0$, то множество $\hat{S} = S' \cap S''$ считается результатом и выдается на печать. В противном случае используются алгоритмы, уточняющие информацию. Операцию пересечения $\cap$ также можно реализовать одним из простейших алгоритмов поиска A^{P2}. Поскольку обьекты множеств S и S_2 поставлены во взаимооднозначное соответствие, то возможна простая проверка по совпадению номеров.

Последовательность преобразования информации при решении этой задачи будет следующей.



(2.3.),

где A^{P2} - алгоритм поиска реализующий пересечение **S'** и **S''**.

Таким образом, анализ информационно распознающих алгоритмов в случае интеграции БД внутри одной персональной ЭВМ показывает, что в модели данных $\mathfrak{M}$ все операции, и БД необходимо задавать как множества. Информационная модель m_2 в этом случае примет вид

$$m_2 = \langle \{T\}, \{\mathfrak{E}(\tilde{A})\}, \{S\}, \{R\} \rangle.$$

2.1.3. Интеграция по функциям обработки и интеграция различных БД внутри одной ЭВМ

По существу, модель m_2 отражает также и этот случай, поскольку неявно кроме интеграции по БД была рассмотрена и интеграция по функциям обработки.

Однако следует сделать ряд замечаний, касающихся интеграции в смысле законченных пакетов. Как правило, каждый пакет работает с вполне определённой структурой данных. Нет проблем обрабатывать данные подобных структур (например, ведение различных БД в рамках одной СУБД.) Но если мы имеем разнотипные БД, появляющиеся в различные время, то нам будет необходимо искать компромисс между специализацией интегрируемых пакетов и их универсализацией. Это может привести к тому, что в рамках одной информационной системы будут функционировать различные пакеты одного функционального предназначения, но настроенные на разные типы данных (например, несколько редакторов: текстовый и графический или даже несколько текстовых.)

Это представляет дополнительные требования к взаимоувязке частей информационной системы и ставит вопрос о её надёжности.

2.1.4. Интеграция по функциям обработки в рамках локальных сетей

Пусть информация сосредоточена в рамках одной или нескольких БД находящихся в одной центральной ЭВМ, соединенной с другими в локальную сеть. Функции обработки данных разнесены по разным ЭВМ.

$\{ R_1^{i1}, R_2^{i2}, \dots, R_p^{ip} \} = R^i$ - подмножество функций обработки разнесенные по ЭВМ, где i_j - номер ЭВМ.

$\{ R_1, R_2, \dots, R_k \} = R$ - подмножество функций обработки выплняемые над данными БД $\{ S_1, S_2, \dots, S_q \}$ в центральной ЭВМ. Естественно, остальных ЭВМ также могут создаватёся некоторое БД в

виде временных файлов. Существенно то, что для получения данных по запросу любая ЭВМ должна обработиться в центральную ЭВМ.

Поскольку данные могут быть получены из одной ЭВМ моменты их обработки должны быть синхронизированы по времени. Таким образом,

$A = A(t)$ и $R = R(t)$. *

При последовательной обработке информации в сети, время выполнения последующих операций не может наступать раньше предыдущех. Разобьем все время обработки)равное временному интервалу $t = [t_1, t_q]$) на подинтервалы равные времени выплнения операции

$[t_{j1}, t_{j2}], \dots, [t_{jg}, t_{jg+1}]$ причём

$t_{j1}=t_1, \quad t_{jg+1}=t_q, \bigcup_{m,k} [t_{jm}, t_{jk}] = [t_1, t_q]$.

Пусть задана некоторая схема Σ выполнения информационно - распознающего алгоритма

$$\Sigma = A^I \cdot A^P \cdot R^{ij} \cdot R^{ik} \cdot A^O.$$

*Где это не будет вызывать противоречий A и R будем отождествлять.

Так как время выполнения операций последовательно, то

$$\bar{E} = A^{I}_{[t1,ti1]} \cdot A^{P}_{[ti1,ti2]} \cdot R^{ij}_{[ti2,ti3]} \cdot R^{ik}_{[ti3,ti4]} \cdot A^{O}_{[ti4,tq]}$$

ЭВМ с номерами i_j и i_k (в интервалы времени $[t_{i2}, t_{i3}]$. $[t_{i3}, t_{i4}]$ соответственно) должны быть готовы к приёму и обработке информации. В противном случае время обработки затянется.

Пример

Пусть из центральной ЭВМ необходимо взять форму письма; секретарь на своей ЭВМ должен эту форму отредактироватьш: а руководитель заполноть содержимое. Тогда

A^{I} - запрос на поиск формы;

A^{P} -поиск формы в БД;

R^{ij} - функции текстового редактора на ЭВМ $_{ij}$;

R^{ik} - также функции текстового редактора на ЭВМ $_{ik}$;

A^{O} - печать письма на центральной ЭВМ.

При последовательном соединении возможны колизии неготовности, когда распределённая обработка данных требует готовности одной из ЭВМ цепи, а она занята другой работой.

При звездообразном соединении возможны колизии доступа, когда к одной БД одновременно осуществляются запросы из разных ЭВМ. При такой форме соединения схемы информационно-распознающих

алгоритмов, будут иметь вид дерева, где вершиной дерева является функции обработки центральной ЭВМ. Правда, в этом случае должен быть реализован механизм постановки запросов в очередь и задача дисциплина обслуживания очередей (например **LIFO - Last Input First Output** или **FIFO - First Input First Output**) или релизована функция запирания транзакции - **LOCK**.

Пусть **F** - множество типов соединения ЭВМ в сеть;

D - множество дисциплин обслуживания очередей центральной машиной.

Тогда информационная модель системы обработки, включающая интеграцию по функциям обработки в рамках локальных сетей примёт вид

$$m_3 = < \{ T(t) \}, \{ \tilde{E} (\tilde{A} (t) \}, S , \{ R (t) \}, D,F, >$$

В рамках такой модели может быть описана схема обработки информации в болшинстве локальных сетей с последовательным соединением.

Здесь **S** подразумевает, что оно может состоять из разных БД, но сосредоточенных в одной ЭВМ.

2.1.5. Интеграция БД в рамках локальных сетей

Рссмотрим случай, когда на различных ЭВМ, соединённых в локальную сеть имеются различные БД, но функции обработки не разделены. Это может иметь место, когда из одной ЭВМ формируется запрос, но заранее неизвестно, где находится информация, требуемая в

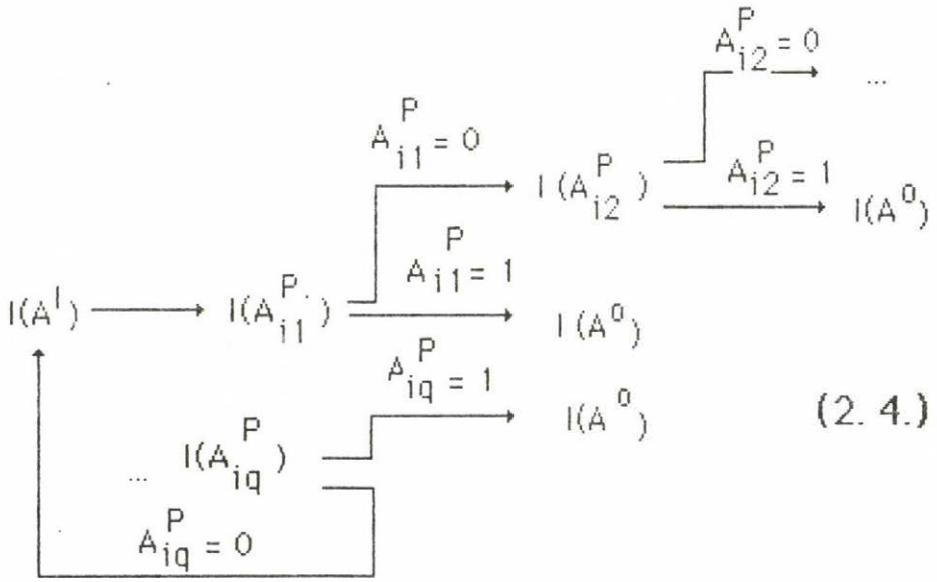
запросе.

Пример

Пусть в ЭВМ, соединённой в локальную сеть хранятся данные о жителях города, причём на каждой ЭВМ хранятся данные одного района. Тогда на вопрос

" где проживает Ремжё Тибор?" необходимо произвести поиск во всех БД.

Обработка информации при таком типе интеграции будет подчинена следующей схеме



(2.4.)

В схеме (2.4.) i_j - номер ЭВМ в локальной сети.

В случае когда в последней ЭВМ (последней доступной БД) требуемая информация не найдена, запрос возвращается для уточнения.

Хотя мы условились, что функции обработки не распределены, но это справедливо лишь в том смысле, что одновременно в локальной сети может выполняться только одно действие относящееся ко всем ЭВМ. Но в

то же время на отдельно взятой ЭВМ могут выполняться действия относящееся к локальной БД (например, обновление, чтение, и т.д.) Так что в общем случае зависимость от времени остается. Дисциплина доступа из-за отсутствия очередей несущественна. Также несущественен тип соединения ЭВМ, поскольку обработка всё равно будет последовательной.

Тогда информационная модель будет представлена в виде

$$m_4 = \langle \{ T(t) \}, \{ \bar{E}(\tilde{A}(t)) \}, \{ S \} \rangle$$

Более сложным представляется случай, когда одна или несколько БД физически разделены между различными машинами.

Тогда при наличии одного действия в локальной сети, возможна конкуренция по обслуживанию разделённых БД. Этот случай мы подробно рассматривать не будем поскольку там существенную роль играет физическая организация, а в настоящей диссертационней работе внимание сосредоточено рассматривается на логическом уровне.

2.1.6. Интеграция по функциям обработки и интеграция БД в рамках локальных сетей

Этот случай по существу представляет собой объединение двух предыдущих и соответствующая информационная модель имеет вид:

$$m_5 = \langle \{ T(t) \}, \{ \tilde{E} (\tilde{A}(t)) \}, \{ S \}, \{ R(t) \}, D, F \rangle$$

Схемы информационно - распознающих алгоритмов в рамках этой модели представляются в виде сети, где в каждом узле стоят: алгоритм, номер ЭВМ, временная синхронизация, вид дисциплины обслуживания; а дуги нагружены условиями перехода от одного алгоритма к другому..

Подчеркнем ещё раз, что представленное исследование носит дескриптивный характер и ставит перед собой следующие задачи:

1. Предложить единнообразный аппарат формального описания различных типов обработки в рамках локальных сетей.

2. Разработать информационные модели соответствующие различным типам обработки.

3. Расширить исследования по информационно-рапознающим алгоритмам в область распределённой обработки данных.

4. Представить методологическую и структурную основу для создания информационной распределённой системы страхования ВНР.

2.2. Информационная модель системы страхования.

Конкретизируем проведенные исследования в случае создания распределённой системы страхования.

2.2.1 Модель страхования уровня страховой конторы

Пусть у нас имеется несколько БД:

S_1 - БД клиентов;

S_2 - БД описания страховок;

S_3 - БД страховка домов;

S_4 - БД страховок автомашин;

S_5 - БД страховок домашних животных;

S_6 - БД страховок дач.

$$S_1 = (a_1^1, ..., a_5^1),$$

где a_1^1 - идентификатор личности;

a_2^1 - фамилия;

a_3^1 - имя;

a_4^1 - адрес жительства;

a_5^1 - профессия.

(Здесь знак равенства подразумевает, что объекта множества S_1 имеют параметры $a_1^1, \ldots a_5^1$)

$$S_2 = (a_1^2, \ldots, a_9^2),$$

где a_1^2 - номер страховой конторы;

a_2^2 - тип контракта;

a_3^2 - место платежей;

a_4^2 - сумма страховки;

a_5^2 - сумма оплаты в год;

a_6^2 - сколько уже оплачено в текущем году;

a_7^2 - сколько осталось оплатить в текущем году;

a_8^2 - дата заключения контракта;

a_9^2 - идентификатор клиента.

$S_3 = (a_1^3, \ldots, a_4^3)$,

где a_1^3 - адрес дома;

a_2^3 - тип дома;

a_3^3 - стоимость дома;

a_4^3 - сумма страховки.

$S_4 = (a_1^4, \ldots, a_5^4)$,

где a_1^4 - номер машины;

a_2^4 - стоимость;

a_3^2 - сумма страховки;

a_4^2 - время эксплуатаяия машины;

a_5^2 - тип машины.

$S_5 = (a_1^5, \dots, a_3^5)$,

где a_1^5 - клична животного;

a_2^5 - стоимость;

a_3^5 - сумма страховки;

$S_6 = (a_1^6, \dots, a_4^6)$,

где a_1^6 - тип дачи;

a_2^6 - адрес;

a_3^6 - стоимост дачи;

a_4^6 - сумма страховки.

Таким образом, мы имеем интеграцию разных БД внутри одной персональной ЭВМ, поскольку в пунктах страхования системы **LATOR** поддерживает иерархическую конфигурацию локальной сети (рис 2.1.)

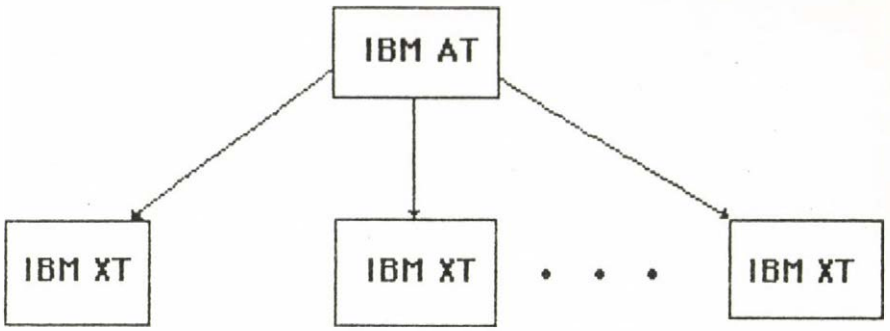


Рис. 2.1. Структура локальной сети

Все базы $S_1, \ldots, S_6$ находятся в ЭВМ **IBM AT.**

Определим функции реализованные на ЭВМ

1. Функции выполняемые на любой ЭВМ:

R_1 - обновление любой из БД;

R_2 - осуществляние страхования;

R_3 - расчёт и выдача компенсаций.

2. Функции выполняемые на **IBM AT.** Будем считать, что **IBM AT** имеет номер 1, а все остальные - **IBM XT** имеют последовательные номера 2,3,4.

R_1^1 - автоматические страховые операции. Например, расчёт, какую сумму каждый клиент должен внести в текущем году.

R_2^1 - проверка БД страховок и печать напоминаний кредиторам;

R_3^1 - изготовление и печать всех необходимых документов;

R_4^1 - ведение архива;

R_5^1 - закрытие дня (включает закрытие всех БД.)

Перечисленные функции не исчерпывает всей работы системы, но вполне достаточны для изучения информационной модели.

Настоящая система страхования описывается информационной моделью

$$m_3 = <\{ T(t) \}, \{ \tilde{E}(\tilde{A}(t)) \}, S, \{ R(t) \}, D, F >,$$

где

$$S = S_1 \cup S_2 .... \cup S_6;$$

$$\{R(t)\} = R_1 \cup R_2 \cup R_3 \cup R_1^1 ... \cup R_5^1.$$

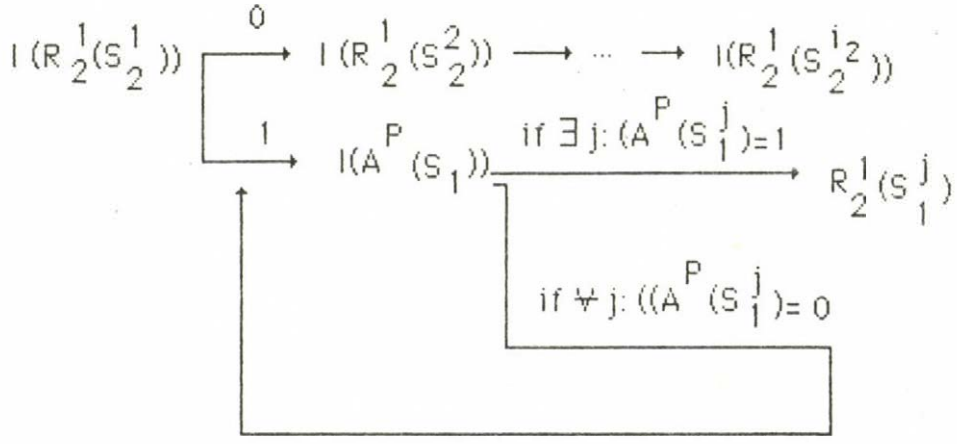
Проанализируем, что в данном случае будут собой представлять алгоритмы поиска и распознавания. Очевидно, что в БД объекты описываются атрибутами разной природы. Так в S_2 есть атрибут a_3^2 -

место платежей, (множество значений которого являются строки символов т.е. оно не являются метрическим) и атрибут a_4^2 - сумма страховки (множество значений которого оринадлеэжит метрическому пространству). Этот факт учитывался при разработке соответствующим алгоритмов системы ABLAK.

Проанализируем несколько содержительных операцией страхования уровня отдельной конторы.

1. Необходимо для всех клиентов вычислить сумму задолженности платежей по страховке в текущем году и подготовить уведомление должникам.

Центральная часть схемы информационно - распознающего алгоритма будет следующая:



Здесь ---^{0}----> обозначает, что, если по страховке $S_2^1 \in S_2$ задолжености нет то обратывается страховка $S_2^{i+1} \in S_2$,

$$\bigcup_{i=1}^{n} S_2^f = S_2 ;$$

—[1]—> обозначает, что если по страховке $S_2^i \in S_2$ задолженность есть, то осуществляется переход к реализации алгоритма поиска по объектам множества S_1 (установливается существует ли такой клиент в БД. Если клиент есть в БД (т.е. $\exists j : (A^p(S_1^j) = 1)$), то производится печать напоминания. Если такой клиент не обнаружен (т.е. $\forall j : (A^p(S_1^j) = 0)$), то производится уточнение.

Мы преднамеренно не расписывали схему информационно-распознающего алгоритма до детальных операций, чтобы не затуманивать понимания в целом.

Такие алгоритмы можно выписать по всем операциям страхования, что даёт возможность лёгкого понимания работа всей системы на логическом уровне.

ГЛАВА 3.

Распределённая система страхования ABLAK

Как отмечалось система ABLAK построена на основе двух программных компонент: сетевой СУБД LATOR и интерактивного языка программирования **LATROQ**. Функции, реализуемые системой ABLAK, по своей сути являются диалоговыми информационно-распознающими алгоритмами, реализованными на основе функций этих системных компонент в рамках информационной модели m_3. Поэтому описание системы ABLAK , по существу, есть описание СУБД **LATOR** и языка **LATROQ**.

3.1. Реализация СУБД **LATOR**

Идея реализация системы **LATOR** несколько отлична от соответствующих комерческих СУБД таких как **dBASE** или **DATAFLEX** поскольку её основная часть размещена в определённой ЭВМ , называемой ЯДРОм (**server**) *. Пользователь общается с этой системой (ЯДРОм) всякий раз, когда он работает со структурами данных поддерживаемые **LATOR** - ом. Это концептуальное отличие подразумевает также ряд других, которые облегчают разработку прикладных программ пользователя.

* Дословный перевод "слуга" но термин "ЯДРО" лучше отражает суть.

Основные части системы **LATOR** следующие:

- данные и связи;
- схемы БД;
- ЯДРО сети (**network server**);
- программы-утилиты;
- интерактивный язык программирования.

Данные и связи

LATOR поддерживает реляционную структуру данных с возможностью установления связей между отдельными отношениями.

Схемы БД

Вначале БД должна быть описана. Число одновременно присуствующих в системе БД логически неограничено. Задание схемы производится не в интерактивной режиме и для её описания необходимо задать:

- имя БД;
- логическую структуру;
- устройство для размещения системы;
- логический порядок символов.

Для логической и физической схем, а также для указания порядка символов создается таблица. Логические и физические схемы являются просто текстовыми файлами. Трансляция логической и

физической схем выполняется программой **SCHEME.**

Утилита **CHARS** обеспечивает переориентацию символов для обработки запросов. Отдельный, текстовый файл создается для документирования. В результате работы вышеупомянутых программ создаётся схема БД в форме, которую может использовать программа-**SERVER.***

Схема БД имеет следующие части:

- таблицу для логической схемы, которая содержит инварианты для структур данных;
- таблицу физической схемы (в ней должны быть определены физические файлы БД);
- таблицу с времязависимой информацией о БД.

После подготовки схемы БД окружение, поддерживающее алгоритмы управления, готово для работы. После этого БД должна быть наполнена данными.

* В дальнейшем мы в зависимости от необходимости будем употреблять термин "ЯДРО" или просто **SERVER.**

ЯДРО сети

Если **LATOR** используется в локальной сети, тогда должен существовать центральный компьютер вместе с программами, выполняемыми в нём, которые управляют сетевой обработкой. Этот компьютер и его программы называются ЯДРОМ-сети. Пользовательские программы на терминалах связываются с ЯДРОМ через сетевой интерфейс. ЯДРО построено таким образом, что никогда не задает вопросов и не ждёт ответов. Оно отвечает только на запросы терминалов. Ответы должны быть правильными, т.е. они должны соответствовать логической схеме БД. После необходимых преобразований ЯДРО выполняет инструкции и отвечает на запросы в соответствии с требованиям терминала, причём каждый терминал имеет свою собственную таблицу для навигации в БД.

Существует два уровня программных средств, обеспечивающих правильное выполнение программ пользователя. Один уровень-это сетевой интерфейс, который переводит инструкции **LATOR**-а в команды системы сетевой коммуникации. Поэтому возможно "погружать" **LATOR** в любое сетевое математическое обеспечение посредством адаптации этого уровня. Другой уровень - языковый интерфейс. Он транслирует инструкции **LATOR**-а и предоставляет возможность написания транзакций на языке окружения. Интерфейс транслирует запросы, сформулированные на логическом уровне к уровню интерпретируемому программами ЯДРА. К настоящему времени интерфейс разработан к языкам **TURBO PASCAL и MICROSOFT-C.**

Если ЯДРО получает команду на запись в БД, то это действие выполняется в ОЗУ и не происходит немедленной записи на диск. Поэтому если даже все транзакции закрыты на терминалах, на

центральной машине для закрытия файлов должна быть вызвана утилита **DBEXIT**.

Проблема конкуренции

В работе с БД очень важно избежать ситуаций "поломки", когда некоторые части операции уже выполняются, но не могут быть закончены. Эту ситуацию очень важно разрешить средствами СУБД для того, чтобы не перегружать программы пользователья. Поэтому все элементарные операции LATOR-а (как для файлов данных, так и для структур БД) сделаны неделымыми (atomic), т.е. они или выполняются полностью или не выполняются вообще.

Восстановление

Для обеспечения восстановлении при инстоляции БД утилитой **JRNCREAT** создаётся системный журнал, который существует в течении всего времени работы с БД.

Существует понятие "точки контроля останова", которая регулируют следующую ситуацию . Имеется ограничение на число открытах транзакций. Если это число превышено, то нельзя вводить новые транзакции. При этом ЯДРО производит ряд действий необходимых для правильного течения событий.

Командный файл (**command file**)

Системный журнал и точки контроля обеспечивают защиту БД от разрушения и от зависания процессора. Но если сломался диск, тогда восстановление возможно только по предыдущей версии БД,

сохраненной на другом диске. Чтобы сделать эту операцию более удобной, существует возможность запросить ЯДРО использовать так называемый командный файл. Он представляет собой тип системного журнала, содержащий все команды обновления, которые ЯДРО получило со времени последней точки контроля. Конечно, командный файл должен хранится на отдельном запоминающем устройстве.

Конкурентный доступ

Может случится, что две транзакции хотят получить одновременный доступ к одной записи. Для предотвращения этой ситуации возможно запереть текущую запись **LOCK** командой. Пока не выполнитя команда **UNLOCK**, другая программа не может запереть эту запись, и она получает доступ только для чтения.

Ситуация "убивающая запирание" может случится, если выполняемая программа хочет запереть одновременно более чем одну запись. ЯДРО не решает автоматически вопрс об "убивании запирания" и это остается на совести программиста, разрабатывающего прикладное обеспечение.

Параметры для работы сети

Работа ЯДРА контролируется рядом параметров, которые устанавливаются на основе ответов на вопросы, задаваемые системой в момент активизации ЯДРА.

Параметры следующие:

- Число терминалов в сети.
- Размер доступного ОЗУ для управления файлами.
- Размер резервируемой области для программ пользователья.
- Переключатель (да/нет), сигнализирущий о том, необходим или нет системный журнал.
- Переключатель для командного файла (да /нет).
- Размер области расширения в случае переполнения системного журнала.
- Число транзакций, после которого формируется контрольная точка.
- Переключатель трассировки. Если он установлен на "да", то все команды, вводимые с терминала, трассируются. Это полезно для отладки.
- Переключатель тестового режима. Если он установлен на "да", то ошибки не вызывают прекращение работы ЯДРА. В противном случае работа останавливается.
- Наибольшее число одновременно используемых "запираний" и переключателей.
- Переключатель, указывающий может ли текущая запись "запереться" автоматически.
- Переключатель дальнейшей работы, показывающий может или нет перезагруженное ЯДРО обновлять существующую БД интрукциями командного файла.

Программы утилиты

Генератор словаря данных (**KONVERT, KONVERC**) . Эти утилиты создают словарь всех имен, используемых в данной БД.

Утилита **JRNCREAT** создаёт файл системного журнала и командный файл.

Утилиты **UTESZT** и **FLASH** позволяют администратору БД использовать инструкции **LATOR**- а непосредственно из терминала.

Утилиты **SAVEREL, LOADREL, RELOAD, CHKONV** позволяют администратору БД реорганизовать структуру БД и заполнять новую БД из старой.

Система запросов

Она поддерживает процедуры, которые используют БД в режиме листания, например, для распечатки.

Особенности:

Записи, определяющие условия, (из файла данных) могут быть собраны вместе.

Собранные записи могут служить дополнительным условием для выбора.

Условия могут быть сформулированы при помощи логических операторов.

Выбранные записи могут просматриваться на дисплее.

Возможна пересортировка выбранных записей.

Запросы должны быть сформулированы на языке **LATROQ**. Программы

в языке **LATROQ** могут быть написаны, отредактированы, транслированы и выполнены подобно тому как это делается в языке **TURBO PASCAL.**

Требования к техническому окружению

Для функцирования системы **LATOR** необходимо:

- **IBM PC XT** или **IBM PC AT** или компьютеры совместимые с ними.
- По крайнер мере один floppy диск.
- Память для программ и БД - около 1 Мбайта.

Для работы в локальной сети необходимо:

- **IBM PC** или совместимые комьпютеры с ОЗУ 256 Кбайт каждый.
- **LAN** интерфейсная карта.

Требования к программному окружению:

- Наличие ОС **MS-DOS** версии 3.1 или выше.
- Наличие сетевого интерфейс,а включающего некоторые типы сетевого математического обеспечения, необходимого для связывания **LATOR** с сетью. Например, для сети **10NET** необходимо присутствие программы **TENNET.EXE** на центральной машине, а на остальных машинах необходимо программа **USERNET.EXE** Более того, независимо от типа сети программа **MEGS.EXE** должна присутствать на всех машинах, использующих **LATOR**.
- Наличие программа **MISS.EXE**. Часто бывает, что БД подключена весь день, но используется редко. Для того чтобы сделать возможным использование центральной машины, когда нет запросов к БД, монитор

БД используется как программа окружения, на фоне которой могут решаться другие задачи. Чтобы такой подход стал возможным, необходим режим разделения времени, что и реализует программа **MISS.EXE**.

• Наличие программ **LOCAL.EXE** и **LOCALS.EXE**. Эти программы используются только на центральной машине. Программа **LOCAL.EXE** обеспечивает работу **LATOR** только на централной ЭВМ. Программа **LOCALS.EXE** совместно с возможностями программы **MISS.EXE** обеспечивает разделение времени на центральной машине для тестирования.

• Наличие программ **LATOR..EXE** или **LHATOR..EXE**. БД управляются программой, входящей в **SERVER**, и это может быть или **LATOR.EXE** или **LHATOR.EXE**.

• Наличие программы **DBEXIT.EXE**. Эта программа служит для правильного закрытия файлов. Она установливается только на центральной машине.

• Необходимы также программы, связывающие пользователя с **LATOR**. Непосредственные команды транслируются утилитой **UTESZT**. Утилита **LATROQ** или другая система интерактивного программирования, служащая для написания приложений, создается на языках **MICROSOFT-C** или **TURBO PASCAL**.

Система **LATOR** размещается на floppy диске в следующих файлах:

MISS.EXE

LOCAL.EXE

LOCALS.EXE

LATOR.EXE

LHATOR.EXE

DBEXIT.EXE

SCHEME.EXE

MEGS.EXE

JRNCREAT.EXE

CHARS.COM

PROC.PAS

PROCINC.PAS

KONVERT.COM

PROC.LIB

KONVERC.EXE

UTESZT.EXE

FLASH.EXE

SAVEREL.COM

LOADREL.COM

RELOAD.EXE

CHKONV.COM

PROCRES.EXE

3.1.1. Описание связей и дисциплина доступа

Записи

В системе используются записи (**record-types**) имеющие имена. Число записей ограничено только объёмом доступной памяти. Множество всех типов записей называется файлом (**logical file**). Имя этого файла идентично типу записи. Имя записи (идентификатор) представляет собой последовательность цифр, начинающуюся с нуля. На файловом уровне можно получить доступ к записи, модифицировать или удалить её, используя её идентификатор.

Запись представляет собой последовательёность полей, каждое из которах имеет имя. Длина записи определяется суммой длин её полей. Длина записи не ограничена, но все записи одного типа должны иметь одинаковую длину. Имеются следующие типы полей:

- СТРОКА - последовательность символов (максимальная длина 480 байт. Но если **LATOR** использует для программирования **PASCAL**, то длина ограничена 255 байтами).
- ЦЕЛОЕ - знаковые целое длиной 16 бит.
- ДЛИННОЕ ЦЕЛОЕ - знаковое целое длинной 32 бита.
- DATA - псевдо-строка, необходимая для представления дат от 1го января 1900 г. до 30го декабря 2076 г. в порядке год-месяц-день.

Было принято решение о нецелесообразности введения других типов полей, поскольку программы могут интерпретировать строки и числа в зависимости от прикладной задачи.

Сортировка файлов. Ключи доступа

К записи возможен доступ по различным ключам. Для этого записи предварительно сортируются относительно введенных ключей. Каждый ключ должен иметь уникальное имя отличное от имени файла. Каждый ключ состоит из ключевых полей (**key-fields**), которые также имеют имя и тип СТРОКА.

Связи

LATOR может устанавливать двоичные связи между записями. На концептуальном уровне тип "связь" является множеством поименованных пар записей. Две записи в связи называются координатами (**coordinates**) связи.

Связи должны быть объявлены симетричными или асиметричными (**symmetric or asymmetric**). Если связь объявлена как симетричная, тогда порядок координат несущественен. С другой стороны, из-за того, что обратная связь также установлена, имя инверсной связи также должно быть дано. Например, если асиметрическая связь имеет имя ДЕТИ, тогда имя обратной связи может быть РОДИТЕЛИ.

Над двоичными связями возможны следующие операции:

- связивание двух записей в соответствии с типом;
- проверять: были ли две записи связаны или нет данных типом связи;
- взять в записи первую координату связи и др.

Существуют автоматические операции над связями:

- если запись удалётся, то удаляется все относящиеся к ней связи;
- если запись вставляется в асиметричную связь, тогда также устанавливается и вставляется обратная связь;
- если связь удаляется, то удаляется и обратная связь.

Навигация по данным

Навигация возможна про помощи обращений. На каждом терминале должна быть активная пеограмма, и каждая из этих програм должна иметь собственную таблицу обращений, управляемую ЯДРОМ. Содержание такой таблицы следующее:

- последовательный номер текущей записи из каждого файла (**record-type**), или число-1, если текущей записи нет в этом файле;
- содержание текущего ряда для каждой записи: последовательные номера координат записей и их типы.

Навигация при помощи последовательных номеров

Навигация внутри файла данных возможно при помощи следующих инструкцией:

- взять первую запись в файле;
- взять последнюю запись в файле;
- взять следующую запись;
- взять предыдущую запись;
- взять одну запись по номеру.

"Первая", "последняя", следующая" относятся к порядку, установленному номерами.

Навигация по ключу

Всегда можно задать последовательность операций для фиксированного файла и ключа. "Первый", "последний", "следующий" связуются для сортировки ключей:

- пусть первая (последняя) запись будет текущей;
- пусть запись для переопределённого ключа будет текущей;
- пусть следующая (или предыдущая) запись будет текущей.

Устанавливается специальный режим доступа по ключу, если заданы определённые "шаблон последовательность символов" и ЯДРО "наблюдает" запись (первая, последняя, следующая, предыдущая). Тогда устанавливается "эквивалентность" между ключём записи и заданным ключём, причём "эквивалентность" понимается в широком смысле : помимо "равенства" (=), возможны "неравенство" (≠), "больше чем" (>) и т.д. . Все это специальным образом устанавливается для последовательности символов.

События в связах

Существует автоматическая сортировка, устанавливаемая на связах, для того чтобы сохранить события с идентичными первыми координатами, закрывая их вместе. Поэтому мы должны говорить о "следующей линии" в связах. События в связах тоже управляются и возможно текущее событие сделать первым, следующим, предыдущим значением связи или первым значением данной (левым или правым) координаты.

Возможна также навигация посредством связей в файле данных.

Доступ к данным

Возможны различные операции в программах пользователя:чтение записи, вставка новой записи или обновление старой и т.д. Нет необходимости читать или записывать сложную запись, т.к. операции задаются над полями.

3.1.2. Схемы БД

БД системы **LATOR** состоит из совокупности структур данных.

БД т.е. файлы, записи, поля, ключи и связи должны быть сначала описаны и определены на логическом уровне (задание логической схемы). Также должно быть определено физическое размещение структуры. Этот шаг называется определением физической схемы.

Имя БД должно быть строкой доступной в **MS-DOS** как имя файла без расширения.

Логическая схема является текстовым файлом и может быть подготовлена любым текстовым редактором. Когда логическая схема подготовлена, она запоминается в файле **MS-DOS** с именем БД и расширением **LOG**. Программа **SCHEME** берёт этот файл и транслирует его в файл с именем **«имя БД». SCH**. Подобно описанной процедуре должен быть создан файл **«имя БД». PHY**.

Описание логической схемы. Элементы логической схемы

Базовым типом данных применяемым в **LATOR** является поле данных. Из полей данных состоит запись. Записи собираются в файл, который имеет имя идентичное записи. С логической точки зрения файл может быть рассмотрен как файл сортирований по ключам. Помимо файлов существуют двоичные связи. Всё это позволяет осуществлять навигацию между файлами.

В соответствие с вышесказанным необходимо определить:

- имя файла данных (типы записей);
- структуру поля каждой записи;
- имена ключей и структур;
- имя связи и структур.

Каждое определение структуры данных состоит из его имени и следующим за ним описанием.

Файл данных описывается своими записями и структурами ключей. Описание записи требует задание спецификации структуры полей. Описание ключа состоит из элементарных ключевых частей.

Описание двоичной связи состоит из определения типа

(симетричная или несиметричная), имен её атрибутов (называемых координатами) и, в случае асиметричной связи, имени инверсии. Связь определяется как асиметричная, если не указывается обратное.

Описание состоит из секций, конец секции помечается ключевым словом типа **endof...** , записанным с новой строки. Некоторые секции содержат подсекции.

Синтаксис

Зарезервированными ключевыми словами являются:

- **KEY-WORD**
- **NAME**
- **NUMBER**
- **INTERVAL.**

Пробел между символами ключевого поля не допускается

KEY-WORDS:

- **VSAM**
- **endofdtfields**
- **endofkeyfields**
- **endofkeys**
- **endofdatafiles**
- **endofrels**
- ***int**

~*long

~*string

~*date

~*sym

~*all.

Ключевые слова не могут быть использованы для имен других структурных единиц.

Ключевые слова (**key-words**) начинающиеся с ~ **endof...** называются определителями секций.

NAMEs - строки длиной не более 6 символов, отличающиеся от ключевых слов. Допускаются буквы английского алфавита, десятичные цифры и символы подчеркивания. Все буквы переводятся в прописные и если имена содержат более 6 символов, то используется только 6.

NUMBER - являются строками десятичных цифр.

INTERVAL - строки, состоящие из двух чисел соединённых "::".

Строка определения может начинаться с пробела.

Только первая строка в описании логической схемы может быть оператором

~ **VSAM LENGTH-OF-FILE.**

Физический файл для ключа индикаций и связей может состоять из двух частей на двух устройствах памяти и длина первой части может быть ограничена. **LENGTH-OF-FILE** - является беззнаковым целым, определяющим длину первой части, соединённой в блоки по 512 символов.

Также могут появляться:

FIELD-TYPE : : = *int

or *long

or *string NNN

or * date.

***int, *long** являются знаковыми целыми имеющие 16 и 32 бита совстветственно; 0<NNN<255 является беззнаковым целым для определения длины строки.

***date** - строка цифр для дат в форме год-месяц-день.

FIELD-NAME : : = NAME

FIELD-DESCR : : FIEL-NAME FIELD-TYPE

or «EMTY»

RECORD-NAME : : = NAME

RECORD-DESCR: : = RECORD-NAME

or RECORD-NAME LENGTH-OF-FILE

SOURCE-DESCR: : = FIELD-NAME

or FIELD-NAME INTERVAL

здесь **INTERVAL** представляет **NNN::MMM** для индикации части строки поля: "от **NNN** го символа до **MMM** го символа"

KEYFIELD-NAME :: = NAME

KEYFIELD-DESCR :: = KEYFIELD-NAME SOURCE-DESCR

KEY-NAME :: = NAME

KEY-DESCR :: = KEY-NAME (CR LF)

KEYFIELD-DESCR ... (CR LF)

/ endofkeyfields (cr lf)

FILE-DESCR :: = RECORD-DESCR (CR LF)

FIELD-DESCR ... (CR LF)

/ endofdtfields (cr lf)

KEY-DESCR ... (cr lf)

/ endofkeys (cr lf)

RELNAME1 :: = NAME

RELNAME2 :: = NAME

FILE1 :: = RECORD-NAME

or *all

or «empty»

FILE2 :: = RECORD-NAME

or *all

or «empty»

REL-DESCR :: = SY-REL-DESCR

or ASY-REL-DESCR

SY-REL-DESCR :: = RELNAME1 * sym FILE1 FILE2

Последняя строка является определением симметричной связи. Если **FILE1** не специфицирован как **«empty»** тогда ***all** понимается так: идентификаторы записи могут попасть из любого определения записи (файла данных.)

ASY-REL-DESCR :: = RELNAME1 RELNAME2 FILE1 FILE2

Этот оператор определяет асинхронную связь. **RELNAME1** и **RELNAME2** (обратная связь) могут отличаться одно от другого. **FILE1** и **FILE2** могут быть одинаковыми. **FILE2** может быть пустым, тогда **FILE1** также пуст.

Описание логической схемы может быть представлено в общем виде

LOGICAL SCHEME :: =/ vsam LENGTH-OF-FILE (optional)

FILE - DESCR ... (cr lf)

/ endofdatafiles (cr lf)

REL-DESCR ... (cr lf)

/ endofrels.

Ограничения:

- Имена полей внутри описания файла и имена ключевых полей внутри описания ключа не могут повторяться.
- Имя типов записей, имен ключей и связей должны быть

уникальными для конкретной БД.

• Имена полей в описаниях структур должны быть уже определены как поля данных в этом же описании файла.

• Имена типов записей в определении связей должны уже появится в описаниях файла.

• Число NNN в описании типа поля не может быть больше 255.

• Первое число NNN в описании ресурсов должно быть меньше, чем MMM.

• Поскольку логическая схема определена как один файл, ключевые слова могут появляется:

- **/ endofdtfields** - по крайнер мере один раз;
- **/ endofdatafiles** - только один раз;
- **/ endofrels** - более чем раз и в указанном порядке.

Структура логической схемы требует:

• Границей первой секцием должно быть **/ endofdtfields**.

• Не должно появиться границ секций между **/ endofdatafiles / endofrels**.

• После **/ endofdtfields** следующей границей может быть только **/endofdtfields** или **/ endofdatafiles**.

• Следующей границей секции после **/ endofkeyfields** может быть только **/ endofkeyfields** или **/ endofkeys**.

• Последней границей секции перед **/ endofkeys** должен быть **/ endofkeyfields**.

• Последней границей секции перед **/ endofdatafiles** может быть

/endofkeys или **/endofdtfields**.

Пример:

```
/vsam LENGTH-OF-FILE
RECORD-NAME
    FIELD-NAME FIELD-TYPE
    FIELD-NAME FIELD-TYPE
    FIELD-NAME FIELD-TYPE
/endofdtfields; the first record has three fields
    KEY-NAME
        KEYFIELD-NAME SOURCE-DESCR
        KEYFIELD-NAME SOURCE-DESCR
        /endofkeyfields
    KEY-NAME
        KEYFIELD-NAME SOURCE-DESCR
        /endofkeyfields
    /endofkeys
    ; there are two ways of accessing the records
    ; of the RECORD-NAME data file.
RECORD-NAME
    FIELD-NAME FIELD-TYPE
    /endofdtfields
    /endofkeys ; records of this file can be accessed
```

```
        ; by their serial numbers only.
    RECORD-NAME
        /endofdtfields
        /endofkeys
        ; this record type nas no data fields at all,
        ; occurrences will have serial numbers only.
    /endofdatafiles
    RELNAME1 RELNAME2 FILE1 FILE2
    RELNAME1 RELNAME2 FILE1 FILE2
    RELNAME1 *sym FILE1 FILE2
    /endofrels ; end of example.
```

Физическая схема

Для того, чтобы система управления файлами идентифицировала имена, они должны быть следующими:

- каталогами;
- индексными файлами для доступа по ключам и связям;
- файлами данных в том порядке, в котором они стоят в логической схеме.

Синтаксис

Первая строка должна содержать имя физического файла в фиксированном порядке.

Первым именем является то, что стоит в каталоге в отдельной строке. Если в логической схеме есть разделитель ~ **vsam** , тогда

следующая строка должна содержать физическое имя так называемого **VSAM** файла.

В описании следующими должны быть имена файлов данных, в том же порядке, в котором они были в логической схеме.

Конечно, имена файлов должны быть доступны **MS-DOS** : они должны иметь расширения и отличаться один от другого, также как и от файлов, существующих в этой директории.

В физической схеме не допускаются комментарии.

Пример

```
a: d:
    c: bcf18ind
    c: index.vsm
    c: adat1.dat        d: folyt1.dat
    c: adat2.dat        d: folyt2.dat
    c: adat3.dat        d:folyt3.dat
    c: adat4.dat        d:folyt4.dat.
```

Требование к размещению в БД

Файлы данных

Логическая длина записи может быть определена как сумма её полей. Физическая запись на 5 байтов длиннее.

Ключи и связи являются обычными структурами записей поэтому их длина может быть посчитана обычным образом. Размер блоков 492

байта.

Файл системного журнала имеет по 524 байта на каждой блок и его максимальный размер определяется в файле параметров.

Командный файл содержит команды обновления, которые могут быть длиной до 492 байт.

Программа **SCHEME**

Транслирует описания логического и физического файла в систему **LATOR** и запускается командой

SCHEME «имя БД»

В результате получаем два файла **«имя БД». SCH** и **«имя БД».PHY**.

Для синтаксической проверки только логической схемы используется команда:

SCHEME «имя БД»↵ C.

Если логическая схема правильная, то команда

SCHEME «имя БД»↵ I

завершает работу.

3.1.3. ЯДРО системы (SERVER)

ЯДРО системы связывает различные программы, такие как **LATOR, LHATOR, LOCAL, LOCALS** и др. Команды, непбходимые для связи интерпретируется непосредственно программой **UTESZT**. ЯДРО также позволяет транслировать прикладные программы пользователя, написанные на одном из включающих языков.

Запуск **SERVER**

Работа с ЯДРОМ возможна или через терминал или через центральную машину. Для запуска **SERVER** должен быть помещен в буфер команд.

Если пользователь не работает с включающим языком и / или **UTESZT** утилитой, а хочет установить простую связь с **LATOR** , то должна выполнятся ассемблерная команда

INT 21 H

с следующим содержанием регистров:

AX = **8000H**

ES:BX = «адрес буфера ответа»

DS:DX = «адрес командного буфера»

CX = «длина команда».

Общие правила синтаксиса **SERVER**

Определяющие слова:

«file» - логическое имя любой определённой записи;

«field» - имя поля в записи (файле);

«key» - устанавливается для любого логического имени ключа;

«key-field-n» -**n**-ое поле ключа;

«relation» - имя связи;

«record number» - строка из десятичных цифр;

«value» - символьная строка, размер поля в записи;

«data» - обычно символьная строка. Может быть именем;

«side» - относится к связи. Может быть "правой" или "левой";

«open string» - строка символов, количество которых определяется параметром;

«relation specifier» <=> / <>> / <!> / < not greater than> / < not less than > / <определение эквивалентности пользователем>.

Прописные и строчные буквы в именах (записях, именах ключей, именах связей и т.д.) также как и ключевые слова **SERVER** не различает.

Номер записи является последовательным номером записи внутри типа записи (файла данных) и используется для идентификации. Номер не изменяется в течении времени существования записи, но при её создании или перезаписи тот же номер может быть присвоен другой записи.

Сортировка по ключам

Вначале записи логически сортируются и группируются в соответсвии с порядком ключевых полей (как специфировано в описании логической схемы).

События (**occurrences**) в связи называются линиями. События, связанные линией связи являются собстственно левой и правой кординатой линии.

Линии в связи сортируются сначала по номеру файла данных в левой координате, затем по последовательному номеру записей внутри файла данных, затем по номеру в правой координате и наконец по последовательному номеру в правой координате.

Команды начинаются с ключевого слова и пробела и не могут нигде больше появляться кроме своей строки (за исключением строки значения поля данных.)

Терминальные символы:

= (эквивалентность) - для имен;

; (точка с занятой) - для значений;

\ (косая черта) - для команд.

Символ повторения "..." - это метасимвол, показывающий, что синтактическая единица повторяется.

Описав синтаксис данных, связей, логические и физические схемы БД и некоторые сведения из внутреннего мира **LATOR** , необходимые для понимания функционирования системы, мы приступаем к описанию элементарных функций, составляющих, по существу, множества операций **{R}**, $\{A^P\}$, $\{A^Z\}$, и операции множества **{T}** в информационной модели. Также отметим, что нами били описаны дисциплины доступа **F** и способ соединения пользователей с центральной ЭВМ - **D**. Таким образом, мы реально описываем все компоненты информационной модели.

Поиск записи данных

Поиск записи, идентифицированный номером записи

\FRIN « файла » = « номер записи » ;

Пример: **\FRIN ufile=123;**

Результат:

DB-ERROR-CODE=0, если такая запись существует. Эта запись становится текущей;

DB-ERROR-CODE=-1, если такой записи не существует.

Поиск первой записи

\FFRN « файла » =

Пример: **\FFRN file=**

Результат:

DB-ERROR-CODE=0, если «файл» не пуст. Первая запись становится текущей.

DB-ERROR-CODE=-1, если «файл» пуст.

Поиск последней записи

\FLRN « файла » =

Пример: **\FLRN file=**

Результат:

DB-ERROR-CODE=0, если «файл» не пуст. Последняя запись становится текущей.

DB-ERROR-CODE=-1, если «файл» пусть.

Поиск следующей записи

\FNRN « файла » =

Пример: **\FNRN file=**

Результат:

DB-ERROR-CODE=0, если «файл» имеет текущую запись и она не последняя. Текущий становится следующая запись. Если в файле нет текущей записи, то такой становиться запись с меньшим номером.

DB-ERROR-CODE=-1, если «файл» пуст или текущая запись последняя.

Поиск предыдущей записи

\FPRN « файла » =

Пример: **\FPRN file=**

Результат аналогичен предыдущему. Только вместо последней рассматривается первая запись.

Поиск записи по ключу

\FRIK « файл » =« ключ » = (« ключ поля » = « значение »;) ...

Пример: **\FRIK file= index=keyf1=1112; keyf2=23**

Результат:

DB-ERROR-CODE=-1, если данная величина ключа больше, чем в каждой существующей записи.

DB-ERROR-CODE=0, и **DB-FLAG=0** если существует по крайней мере одна запись с указанной величиной ключа. Текущей становится запись с меньшим номером.

DB-ERROR-CODE=0, и **DB-FLAG=1,** если не существует в «файле» записи с данным ключем, но существует по крайней мере одна со значением ключа больше чем указанный. Текущей становиться запись с наименьшим номером.

Нахождение первой записи по ключу

\FFRK « файл » =« ключ » =

Пример: **\FFRK file=index=**

Результат:

DB-ERROR-CODE=0, если «файл» не пуст. Текущей становится первая запись, соответствующая указанной структуре ключа. Если таких записей несколько, то текущей становится запись с наименьшим ключем.

DB-ERROR-CODE=-1, если «файл» пуст.

Нахождение последней записи по ключу

\FLRK « файл » =« ключ » =

Пример: **\FLRK file=index=**

Результат аналогичен предыдущему.

Нахождение следующей записи по ключу

\FNRK « файл » =« ключ » =

Пример: **\FNRK file=index=**

Результат:

DB-ERROR-CODE=-1, если «файл» имел уже текущую запись (даже удалённую) и есть следующая запись (по крайней мере одна), соответствующая ключу доступа. Запись с наименьшим номером становится текущей. Если «файл» не имеет текущей записи, но не пуст, тогда первая запись соответствующая ключу доступа, становится текущей.

DB-FLAG=-1, если изменеие события требует изменения величены ключа или если «файл» не имеет текущей записи.

DB-FLAG=-1, если новая текущая запись имеет тот же ключ, что в запросе.

Нахождение предыдущей записи по ключу

\FPRK « файл » =« ключ » =

Пример: **\FPRK file=index=**

Результат:

DB-ERROR-CODE=-1, если нет записей в «файле» или текущая запись

(даже удалённая, но сохраняющая точку сылки) является первой отсортированной по заданному ключу.

DB-ERROR-CODE=0, если файл имеет текущую запись (даже удалённую) и существует предыдущая, соответствующая «ключу». Эта последняя становится текущей и в случае, если файл не имеет текущей записи, но не пуст.

DB-FLAG=0, если новая текущая запись имеет такой же «ключ», что и в запросе.

DB-FLAG=0, если новая текущая запись имеет «ключ» отличный от заданного или если не существует текущей записи.

Нахождения первой записи среди подобнах

**\FFRS « файл » = « ключ » = (« поле ключа » « спецификатор связи» « значение »;) ... **

Пример:

\FFRS file= perkey=indf1«Smithye;indf1»smit;зарплата!200;

Результат:

SERVER сравнивает величину, заданную в команде, с записями в файле данных и первую запись, для которой выполнится условие делает текущей. Если условий больше одного, то они рассматриваются как

соединённые логическим "И".

DB-ERROR-CODE= 0

DB-ERROR-CODE = -1, если в «файле» нет записи, которая соответсвует условию или «файл» пуст. Заметим, что значения **DB-FLAG** не рассматривается.

Следующие перечисленное команды реализуются в соответствии с вычшеизложенной идеологией. Мы их просто пречислим:

1. Нахождение последней записи среди подобных.
2. Нахождение следующей записи среди подобных.
3. Нахождение предыдущий записи среди подобных.
4. Нахождение первой линии в связи.
5. Нахождение последней линии в связи.
6. Нахождение следующей линии в связи.
7. Нахождение следующей линии в симетричной связи.
8. Нахождение предыдущей линии в связи.
9. Нахождение предыдущей линии в симетричной связи.
10. Нахождение атрибутов в связи.

Давайте теперь осмыслым, чем же являются перечисленные команды в смысле функционирования информационной модели.

При рассмотрении структур информационно-расспознающих алгоритмов мы оперировали понятием информации: $I(A^P)$, $I(A^Z)$, $I(A^I)$, и т.д. не задумываясь об её структуре. **LATOR** поддерживает достаточно сложные структуры данных, состоящее из файлов, записей, полей, ключей. Это необходимо, чтобы получить доступ к записям, связям, и т.д. Необходимо организовать элементарный поиск (реализовать ряд различных алгоритмов A^P), по разным условиям. Эти алгоритмы и были

описаны выше. Они входят в информационную модель и также соединяются в информационно- распознающие алгоритмы.

В системе **LATOR** есть группа элементарных функций (команд), которые отвечают за организацию БД (т.е. строят информацию, с которой работают алгоритмы более высокого уровня.)

Команды СУБД уровня запись- файл

Удаление текущей записи в файле

**\DCRF « файл » = **

Пример: **\DCRF file= **

Результат:

DB-ERROR-CODE = -1 , если текущей записи в «файле» нет.

DB-ERROR-CODE = 0 , в противном случае.

DB-LOCK=1 , если запись находится в "запертом" состоянии и команда не может быть выполнена.

DB-LOCK=1 , в противном случае.

Размещение записи в файле

\STRF « файл »= (« поле »=« значение »;) ...

Пример:

\STRF file=name=Smith;зарплата=200;

Результат:

DB-ERROR-CODE=0, если запись вставлена.

Связывание записей в связь

\CORR « файл » = (« файл_1 » = « номер записи_1 » ; « файл_2 » =« номер_записи_2 »;)

Пример:

\CORR connex=ufileA=23;ufileB=46;

Результат:

DB-ERROR-CODE=0, если требуемая линия успешно вставлена в "связь";

DB-ERROR-CODE=-1, если указанные записи не существуют.

Поскольку последующие команды этой группы имеют сходную идеологию выполнения, мы их просто перечислим.

1. Освобождение записей от связей.
2. Проверка записей в связи.
3. Модифицировать текущую запись в файле.
4. Взять текущую запись из файла.
5. "Запереть" текущую запись в файле.

6. Освободить каждую "запертую" запись.

7. Освободить файл.

8. Взять ключ текущей записи.

9. Взять текущую линию в связи.

10. Найти номер текущей записи в файле.

Глобальные команды

Следующий уровень команд в **SERVER** представляют команды, выполняемые на основе двух предыдущих уровней. Таким образом, каждая команда представляет собой особый информационно-распознающий алгоритм, состоящий из композиции команд (функций) низшего уровня.

Читать данные в область пользователя

\RDUS

Результат: Копия области пользователя помещается в **DB-IN-PARS.**

Записать из области пользователя

\WRUS= « открытая строка»;

Пример: **\WRUS=gff 57 not 0000jjj;**

Результат:

«открытая строка» выбирается из области пользователя. Если она короче чем заданный размер области пользователя, то остаток не изменяется.

Очистить область пользователя

\CLUS

Результат:
Область пользователя заполняется пробелами.

Зарегестрировать транзакцию.

\LGIN « данные »

Здесь данные являются строкой, данной пользователем для идентификации транзакции инициированной командой **\LGIN,** и которую могут прервать следующие команды: **\LOFF, \TEND, \LEND, \LFSH.** Терминал, с которого поступила команда **\LGIN,** регистрируется локальной сетью и становится изветсным **SERVER.** Поэтому **\LGIN** не означает нового пользователя.

Результат:

DB-ERROR-CODE = -1, если доступ к БД ещё невозможен, в противном случае параметр равен нулю.

Выключить транзакцию

\LOFF

Результат:

Записи освобождаются. События сохраняются для пользовательской последовательности транзакций.

Транзакция **END**

\TEND

Результат:

Это вид команды **LOFF.** Она используется для того, чтобы при необходимости организовать контрольную точку. При этом сессия продолжается с другой транзакцией. Поэтому события, "запирания" и "семафоры" сохраняются.

Конец сессии **LATOR**

\LEND

Результат:

Конец сессии. Ничто не сохраняется: записи освобождаются, семафоры становятся свободным и все события теряются.

Имется ещё ряд глобальных операций выполняемых по такой же идеологии.

1. Глобальные удаление файла.
2. Запирание "семафора".
3. Реализация "семафора".
4. Сбор статистики по файлу.
5. Глобальный выход.

Список синтактических ошибок, выдаваемых при работе **SERVER** приведены в приложении 1.

Список параметров и сообщений **SERVER** приведен в приложении 2.

Для нормального закрытия системы **LATOR** необходимо, чтобы выполнялась программа **DBEXIT.EXE.**

Таким образом, мы кратко описали основные команды работы с БД на системном уровне, которые входят как составные части в информационно- распознающие алгоритмы различных уровней обработки.

3.1.4. Программы - утилиты

Программы утилиты выполняют вспомагательную роль, но без и их функционирование системы **LATOR** невозможно. Для понимания функций программ - утилит в рамках информационной модели рассмотрим простейший информационно-распознающий алгоритм со следующей схемой.

$$\tilde{S} = I(A^I) \bullet I(A^P) \bullet I(A^M) \bullet I(A^O)$$

Алгоритм поиска работает с одной стороны с информационным описанием объекта $\tilde{S}$, полученное алгоритмом A^I, а с другой стороны с множеством **S** - БД, имеющей свою сложную внутреннюю структуру. Вопросы, связанные с созданием БД, её реорганизацией и т.д. не рассматривались при анализе модели, однако они необходимы при функционировании реальной системы. Программы - утилиты обеспечивают этот процесс.

Пусть некоторый алгоритм запущен с некоторого терминала, но должен быть отложен из-за конкурентной работы другого алгоритма. Тогда программы-утилиты ведут системной журнал для откладывания процесса, т.е. обеспечивают дисциплину доступа из **{F}**.

Таким образом, можно сказать, что программы - утилиты обеспечивают:

1. Введение БД - **{S}** внутри информационной модели.

2. Одновременное введение многих информационно- распознающих алгоритмов (например, разрешение конфликта между терминалами, запуск и останов некоторых процессов.)

Программы - утилиты также представляют собой набор функций, который может быть включен в модель, но это ничего не добавит в семантику и усложнит восприятие.

Охарактеризуем кратко программы-утилиты.

KONVERT и KONVERC

Эти программы создают словарь БД соответственно для **TURBO PASCAL** и **Microsoft-C.**

JRNCREAT

Эта утилита создаёт файл системного журнала.

CHARS

При помощи этой утилиты предоставляется возможность переопределения порядка символов. Базовый порядок соответствует порядку кодов **ASCII.** Новый порядок задается в файле

«имя БД». CHR.,

который создаётся программой **CHARS.**

UTESZT

С помощью этой утилиты предоставляется возможность давать команды **SERVER**-у непосредственно с дисплея и получать ответ.

SAVEREL, LOADREL, RELOAD, CHKONV

Эти программы позволяют реорганизовать БД.

3.1.5. Интерактивный язык программирования- **LATROQ**

Интерактивный язык программирования **LATROQ** системы **LATOR** представляет собой обычной **PASCAL**-подобный язык с некоторыми особенностями определяемыми спецификой СУБД. Любой вид поиска, печатания ответов и сложные запросы могут быть выполнены на **LATROQ**. Более того он способен обновлять БД (размещать, модифицировать, удалять данные и т.д.)

Используя **LATROQ** легко установить связи между БД **LATOR** и другими системами (при помощи функций экспорта и импорта).

Язык состоит из трёх основных частей: языка редактора, компилятора, и интерпретатора для выплнения скомпилированных программ.

С точки зрения информационно - распознающих алгоритмов с помощю **LATROQ** можно выполнять следующие функции:

1. Генерировать схемы информационно - распознающих алгоритмов.
2. Транслировать их в вид, который понятен СУБД **LATOR**.
3. Выполнять информационно - распознающие алгоритмы.

Кратко опишем систему программирования **LATROQ**.

Установка LATROQ

Система работает в **MS-DOS** совместимых операционных системах. Для работы с сиетемой необходимо наличие следующие файлов:

LATROQ.COM - главная программа;

LATROQ.000 - оверлейные файлы;

LATROQ.001

LATROQ.HLP - краткое описание команд редактора;

LATROQ.MSG - список сообщений об ошибок компиляции.

Структура используемой БД всегда должна находится в файле схемы БД. Его структура несколько отлична от структуры схемы БД **LATOR**, но может быть получена из неё утилитой **QSEMA**. Полная спецификация схемы **LATROQ** должна быть дана в первой линии префиксного файла **LATROQ.PRF** или, если такого файла в текущей поддиректори нет, спецификация схемы должна быть напечатана после подсказки

´A séma neve:´

Порядок действия при установке системы следующий:

LATROQ - вход в главное меню;

LATROQ source-file-spec e - - загрузка и редактирование файла;

LATROQ source-file-spec C - первый проход компиляции;

LATROQ source-file-spec - второй проход компиляции;

LATROQ batch-file-spec - окончание компиляции и размещение кодов как списка в batch файле.

Использование редактора

Команды радактора:

1. Команды движения курсора;
2. Команды вставки и удаления;
3. Команды управления блоком;
4. Смешанные команды.

Команды движения курсора

Движение курсора	Ключи
символ влево	Left
символ вправо	Right
слово влево	Ctrl + Left
слово вправо	Ctrl + Right
линия вверх	Up
линия вниз	Down
на левый конец линии	Home
на правой конец линии	End
начало следующей линии	Enter
в начало экрана	Ctrl + Home
в конец экрана	Ctrl + End
страница вверх	Pg + Dn

страница вниз	Ctrl + PgUp
начало текста	Ctrl+PgDn

Команды вставки и удаления

Действия	Ключи	Имя
вставить режим да / нет	Ins	
вставить линию:		
перед текущей линией	Ctrl + N	insl
после последней линии	Enter	
удалить символ:		
над курсорам	Del	
слева от курсора	Backspace	
удалить линию	Alt + D	dell.

Команды управления блоком

Управления	Ключ	Имя
Пометить блок :		
начало/ конец	F3	bldef
Пометить линию	F4	linebl
копировать блок	F5	blcopy
двигать блок	F7	blmove
удалить блок	F8	bldel

Смешанные команды

Действие	Ключ
помощь	F1
интерактивный режим	F2
конец редактирования	F10
установить окружение текста	Shift + F1
цвет текста	Shift + F2
ввести поле окружения	Shift + F3
ввести цвет поля	Shift + F4
индикация ошибок в полях	Shift + F5.

Интерактивный режим редактора

Интерактивный режим редактора помогает пользователям написать правильную программу, даже, без глубоко знания синтаксиса **LATROQ** и структуы БД. На всех этапах редактирования имеются подсказки, которые позволяют пользователю вести эффективное программирование.

Выполнение програм **LATROQ**

Обычно, интерпретация осуществляется следующей командой

[путь] RUN batch-file [profile],

где **путь** указывает поддиректорию содержащую интерпретатор, **batch-file** - список прекомпилированных программ **LATROQ**, которые должны выполнятся в этой сессии, второй параметр может установить **profile** вместо установливаемого по умолчанию **LATROQ.PRF**.

Определение языка **LATROQ**

Мы не будем давать подробного описание языка, а просто перечислим некоторые основные понятия.

Идентификаторы используются для указания переменным, меток и объектов из БД. Идентификатор состоит из буквы и следующей за ней любой комбинации букв, цифр и нижних подчеркиваний (до 80 символов), но только первые 6 распознаются.

Числа - константы от -32768 до 32767. Перед положительными числами + не ставится.

Литералы - строки констант заключённых в двойные скобки.

Коментарии - последовательность символов заключенных в фигурные скобки.

Структура программы

LATROQ программы имеют описательную часть, следующую за выполняемой частью.

Типы данных **LATROQ** программы работают со следующими типами данных:

целые;
строковые;
файлы;
списки;
битовые карты (bitmap);
метки.

Объявляние типов переменных производится в соответствии со списком

RNT	идентификатор;
STRING	идентификатор [размер];
FILE	идентификатор : =' **file spec** ';
LIST	идентификатор **FOR data-file-name**[S_1,S_2, ...];
BITMAP	идентификатор **FOR data-file-name**;
LABEL	идентификатор;

где S_1,S_2, ... строковые переменные.

Выражения

Простейшими выражениями являются:

- идентификаторы целых, строковых и **bitmap** переменных;
- имена полей и ключевых полей (также используемое как целые или строковые переменные);
- идентификаторы стандартных функций.

Новые выражения могут быть построены из других посредством операторов.

Операторы делятся на пять категорий в зависимости от порядка выполнения.

Операторы и функции мы приводить не будем, поскольку они идентичны операторам языка **PASCAL**.

3.2. Реализация системы ABLAK

В филиалах страховых обществ Венгрии работает информационная система ABLAK. В каждом из 400 филиалов находится локальная вычислительная сеть. Локальные вычислительные сети состоят из персональных вычислительных машин типа IBM PC AT и IBM PC XT. Типичная конфигурация локальной сети:

1 микромашина IBM AT (центральная машина):

оперативная память 2 Мбайта;

2 диска с мощностью 30 Мбайтов;

2 устройства для гибких дисков;

1 печатающее устройство;

1 устройство для архивации (streamer).

3 микромашины IBM XT (рабочие места);

оперативная память 640 Кбайтов;

2 устройства для гибких дисков.

Микромашины сделаны в Италии фирмой OLIVETTI. Система управления локальной сети : FOX-OLIVETTI 10NET.

На диске центральной машины расположена БД. В качестве системы управления БДвыбран **LATOR**. При создании информационной системы ABLAK , состоящей из 130 программ, использованы возможности системы **LATOR**.

Системой ABLAK пользуются работники страховых обществ, поэтому она построена так, чтобы и неспециалисты в вычислительной технике смогли работать на ней.

Общая структура системы ABLAK

В системе ABLAK все микромашины (и центральная машина и рабочие места) используются для работы. Пользовательские программы, запущенные на разных машинах, действуют независимо друг от друга, одновременно каждая может сделать обновление и поиск в базе данных. Координация осуществляется с помощью IONET и LATOR.

Функции ABLAK разделяются на две части:

- **Функции, вызываемые в диалоговом режиме**
  (действуют на любой машине):
  обновление базы ;
  непосредственая таксация;
  оформление выдачи компенсации;
  и.т.д.

- **Функции, выполняемые в пакетном режиме**
  (действуют на центральной машине):
  автоматическая таксация;
  изготовление распечаток;
  архив данных;
  закрытие дня;
  и.т.д.

Любая функция диалога выбирается из набора функций (меню), и эта функция либо запускает программу, выполняющую некоторую задачу, либо показывает новый набор функций (подменю).

Функции, работающие в пакетном режиме берут на себя управление всей БД. Во время их действия функции диалога не могут быть вызваны.

Способы архивации данных:

- содержание БД регулярно (каждые 2-3 дня) переписывается на магнитную ленту, чтобы сохранить её от разрушения;

- неактуальные данные регулярно (раз в месяц, раз в год) исключаются из БД.

Достоверность базы автоматически обеспечивается системой управления **LATOR**.

Обработка данных в локальной сети создаёт проблему конкуррентного доступа к данным для обновления. Возможность блокировки данных решает эту проблему.

Структура данных в БД системы ABLAK

В описании структуры данных чётко задаются объекты (наименование, атрибуты, ключи доступа), и реляции между объектами.

Список объектов (непольное перечисление):

- тип страховки;
- описание страховки;
- описание страховиков;
- описание страховых агентов;
- описание ущербов;
- расчёт;
- данные филиалов;
- данные администраторов;
- календарь;

  и т.д.

Список связей (непольное перечисление)

- тип страховки-страховка;
- страховка-страховик;
- страховка-страховый агент;
- страховка-ущерб;
- календарь-плата за страховку

  и т.д.

Связи играют большую роль в поисках данных, далее они дают естественные группировки данных, которые используется при распечатке, при архивации и т.д.

Число объектов: 42;

Число связей: 25;

Число страховок: 25000 (5000-70000).

Функции системы ABLAK

Полное число функций, вызывающих определённую операцию над страховками - 80. (Каждая функция является отдельным информационно - распознающим алгоритмом.)

Функции диалога разделяются на следующие группы:

- операции над страховками:

  приём;
  модификация;
  передача в другой филиал;
  приём из другого филиала.

- таксация;

- регистрация данных ущербов;

- регистрация данных аварий автомашин.

Различные типы страховок обладают различными параметрами, правилами приёма и таксации. Эти функции осуществляются

отдельными программами, и название программ сохраняется в описании типа страховки. Пример описания типа страховки:

320ст	•название типа
.....	•атрибут
.....	•атрибут
.....	•атрибут
прм320ст	•место названия программы приёма
мод320ст	•место названия программы модификации
ткс320ст	•место названия программы таксации.

Итоги

Внедрение системы ABLAK во всех филиалах предоставляет много преимуществ и сопровождается рядом трудностями.

Трудности:

- необходимо обеспечить, чтобы тот же самый экземпляр системы ABLAK работал во всех филиалах;
- необходимо стандартизировать правила, действия;
- данные 10 миллионов страховок следует внести в вычислительные машины за короткое время.

Преимущества:

- система ABLAK освобождает работников филиала от привычной канцелярской работы и предоставляет новые возможности в страховании;

- система обеспечивает данные управлению страховыми обществами;

- система обеспечивает данные для разработки новых типов страховок.

3.3. Оценка эффективности СУБД **LATOR** и системы **ABLAK**

Основные достоинства СУБД **LATOR**:

1. Время доступа к записьям меньше чем в СУБД **DATAFLEX** в 3-5 раз и 5-8 раз **dBASE III +**.

2. Произвольный доступ не только к записям но и к полям отдельных записей (при том же времени доступа).

3. Расширенная навигация.

4. Удобный интерфейс с пользовательскими программами, написанными на языках **Microsoft-C** и **Turbo PASCAL**.

5. Автоматическое резервирование данных, возможность автоматического и пакетного восстановления БД, что обеспечивает высокую надежность СУБД.

6. Возможность быстрого обнаружения причин разрушения БД и средства её быстрого восстановления.

7. Неограниченное число используемых одновременно БД, ключей, возможность использования составных ключей.

8. Язык **LATROQ** удобный язык **(QUERY by EXAMPLE)** манипуляции СУБД так и рядовому пользователю-программисту (язык имеет удобный редактор и автоматическую интерпретацию команд с диагностикой ошибок).

9. Наличие специального программного слоя настраиваемого на конкретную конфигурацию сети, что делает возможным применение СУБД **LATOR** практически независимым от реализации локальной сети.

10. Ряд специальных экранных эффектов, предусмотренных в СУБД, позволяет осуществлять работу с экраном в режиме, удобном для пользователя.

11. СУБД ЛАТОР написана на языке **Microsoft-C**, что делает её независимый от применяемого типа ЭВМ (система реализуется на ПЭВМ серии **PS/2**.)

Выводы и результаты

В диссертационной работе получены следующие основные результаты:

1. Разработана информационная модель распределённой обработки информации.

2. Изучены вопросы интеграции функций и данных в сетях и предложены соответстующие информационные модели.

3. Разработана сетевая СУБД **LATOR** (включающая: средства сетевого управления СУБД, средства описания БД, интерактивной язык программирования).

4. На основе СУБД **LATOR** разработана система страхования в ВНР **ABLAK**, реализующая информационно - распознающие алгоритмы предложенной модели.

Развитие работы в направлении более глубокого изучения информационной модели распределённой обработки информации позволит создавать разнообразные прикладные системы.

Список основной использованной литературы

1. Алексеенко Е.А., Довгялло А.М., Формализованная модель диалоговой программно-технической системы. - Кибернетика, 1980, №4, с.35-40.

2. Брябрин В.М., Поспелов Д.А., Проблемы построения диалоговых систем для общения с системами искусственного интеллекта. - Материалы семинара **Человеко-машинные системы**. - М.: Знание, 1977,с.3-18.

3. Видоменко В.П., О структурной интерпретации языков информационно-логического программирования. - Программирование, 1980,№ 5, с.56-62.

4. Гладун В. П., Эвристический поиск в сложных средах. - Киев: Наукова думка, 1977. - 165 с.

5. Глушков В.М., Стогний А.А., Афанасьев В.Н., Автоматизированные информационные системы. - М.: Знание, 1973. - 23 с.

6. Деметрович Я., Ханнак Л., Ремжё Т., Урбански Ф., ЛАТОР-профессиональная система управления с базой данных для локальных сетей. - Международная конференция по искусственному интеллекту, Пхеньян,КНДР, 1988.-с.225-247.

7. Деметрович Я., Ханнак Л., Ремжё Т., Урбански Ф., Об информационной системе ABLAK. - Международная конференция по искусственному интеллекту, Пхеньян, КНДР, 1988. - с. 247-252.

8. Довгялло А.М., Диалог пользователя и ЭВМ и место средств искусственного интеллекта в его реализации. - Кибернетика, 1979, № 2, с.102-108.

9. Журавлёв Ю.И., Об алгебраическом подходе к решению задач распознавания или классификация. - В кн. : Проблемы кибернетики, М., № 33, с.5-62.

10. Зайченко Ю.П., Гонта Ю.В., Структурная оптимизация сетей ЭВМ. Киев: Техника, 1986., 168. с.

11. Иванченко А.Ф., Колинько В.В., Кондратьев А.И., Полумиенко С.К., О языке запросов в информационно-раснознающих системах. - Докл. и сообщ. третьей школы-семинара: Интерактивные системы. - Боржоми, 1981 с.243-245.

12. Калиниченко Л.А., Методы и средства интеграции неоднородных баз данных. - М.: Наука, 1983. - 424 с.

13. Классификация и кластер. - под редакц. Журавлёва Ю.И. - М.: Мир, 1980. - 385. с.

14. Криницкий Н.А., Криницкий В.Н., Степанченко Д.А. - О структуре информационных систем. - Программирование, 1975, № 2, с. 3-14.

15. Криницкий Н.А., Миронов Г.А., Фролов Г.Д., Автоматизированные информационные системы. - М.: Наука, 1982. -381. с.

16. Колинько В.В., Кондратьев А.И., О свойстве алгоритмов синтеза начальной информации в информационно-распознающих системах. - ДАН АН УССР, 1981, № 11, с. 81-83.

17. Колинько В.В., Кондратьев А.И., О сценариях диалогов в информационно-распознающих системах. - В. Кн.: Банки данных и информационно-поисковые системы. - Киев: 1980. с. 117-119.

18. Колинько В.В., Кондратьев А.И., Об одном классе алгоритмов синтеза объектов. - ДАН АН УССР, 1981, № 10, с. 75-77.

19. Колынько В.В., Специализированные средства обработки фактографической информации на основе диалоговых информационно-распознающих алгоритмов, автореф. дис. раб., Киев, 1983. - 16. с.

20. Ланкастер Ф., Информационно-поисковые системы. - М.: Мир, 1972.-308.с.

21. Ларионов А.М., Майоров С.А., Новиков Г.И., Вычислительные комплексы, системы и сети. - Ленинград: Энергоатомиздат. - с. 288.

22. Минский М., Фреймы для представления знаний. - М.: Энергия, 1979.-151.с.

23. Мартин Дж., Организация баз данных в вычислительных системах. - М.: Мир, 1980. - 662. с.

24. Перевозчикова О.Л., Ющенко Е.Л., Системы диалогового решения задач на ЭВМ. - Киев: Наукова думка, 1986. - 264 с.

25. Прангишвили И.В., Распредельные микропроцессорные системы обработки данных и управления, В кн.: Информатика, управление, вычислительная техника. - М.: Машиностроение, 1987, с. 44-59.

26. Редько В.Н., Семантическе структуры программ. - Программирование, 1981, № 1, с. 3-19.

27. Ремжё Т, Проблемы интеграции в базах данных филиалов страховых учреждениях, MTA SZTAKI Közlemények, 1988. 203/1988. pp 121-135.

28. Самойленко, С.И., Сети ЭВМ, М.: Наука, 1986, с. 108-110.

29. Стогний А.А., Кондратьев А.И., Информационные системы в управлении. - Киев: Знание, 1980. - 47. с.

30. Стогний А.А., Об основных принципах построения автоматизированных информационных систем. - УСиМ, 1972, № 2, с. 5-10.

31. Стогний А.А., Кондратьев А.И., О стратегическом представлении знаний в АИС. - ДАН АН УССР, 1980, № 5, с. 77-79.

32. Сэлтон Г., Автоматическая обработка, хранение и поиск информации. - М.: Советское радио, 1973. - 432. с.

33. Соколов А.В., Информационно-поисковые системы. - М.: Радио и Связь, 1981. - 151. с.

34. Сухарь М.Я., Хенчей Г., О выделении в информационном массиве подмножества объектов, похожих на данный объект, Вопросы кибернетики, Москва, 1987. - с. 162-168.

35. Тиори Т., Фрай Дж., Проектирование структур баз данных. - М.: Мир, 1985.,с.607.

36. Хенчей Г., Ассоциативно-структурный анализ булевых данных с применением монотонных систем, Автоматика и Телемеханика, Москва, 1987.-с.137-141.

37. Черный А.И., Введение в теорию информационного поиска. - М.: Наука, 1979. - 238. с.

38. Ющенко Е.Л., Перевозчикова О.Л., Развитие языков программирования и диалоговых систем в СССР. - Кибернетика, 1976, № 6,с. 16-33.

39. Информационные системы общего назначения. Под ред. Ющенко Е.Л. - М.: Статистика, 1975. - 430. с.

40. Bridges S., Low-Cost Local Area Networks, John Wiley & Sons Ltd, England, 1986, pp 188.

41. Brookshear J.G., Computer Science: An Overview, Addison-Wesley Publ. Group, Amsterdam, 1988, pp 550.

42. Brumm P., The Micro to Mainframe Connection, John Wiley & Sons Ltd, England, 1986, pp. 222.

43. Cheong V.E., Hirscheim R.A., Local Area Networks, John Wiley & Sons Ltd, England, 1983, pp 208.

44. Copelli S., Gottlob G., Implementation of a Distributed File System on a Geographic Network of Personal Computers, In: Distributed Data Sharing Systems (Proc. of the Third International Seminar on Distributed Data Sharing Systems, Parma, Italy) Ed.: Schreiber F.A., Litwin W., North-Holland, 1985., pp 203-221.

45. Crichlow J.M., An Introduction to Distributed and Parallel Computing, Prentice Hall International, England, 1987, pp 256.

46. Dallas I.N., Spratt E.B., Issues in LAN Management (Proc. of the IFIP TC6 WG6.4 A Workshop on LAN Management, FRG), North-Holland, 1986, pp 398.

47. Demetrovics J., Hannák L., Remzsö T., Urbánszki F., LATOR - a Database Management System for Local Area Network, In: Artificial Intelligence III., Methodology, systems applications (Proc. of the Third International Conference on Artificial Intelligence, Varna, 1988.) Ed.: O'Shea T., Sgurev, V., North - Holland, 1988. pp 347-355.

48. Doswell A., Office Automation, John Wiley & Sons, England, 1983, pp 294.

49. Friedman A., Cornford D., Computer Systems Development - An Historical Analysis, John Wiley & Sons Ltd, England, 1986, pp 218.

50. Gandy M., Choosing a Local Area Network, John Wiley & Sons, England, 1987, pp 106.

51. Garland J.L., How to Develop Business Information Systems for End Users, John Wiley & Sons, England, 1986, pp 254.

52. Lafuente J.M., Gries D., Language facilities for programming user-computer dialogues. - IBM Journal Research and Developement, 1978, Vol 22, Nº 2, p. 145-158.

53. LAN Operating System Report, NOVELL, INC., Orem, Utah, 1986.

54. Andó Gy., Gyepesi Gy., Hannák L., Remzsö T., LATOR - a Database Management system for Local Area Networks. Basic Concepts. MTA SZTAKI Budapest, 1987.

55. Andó Gy., Gyepesi Gy., Hannák L., Remzsö T., LATOR - a Database Management system for Local Area Networks. The Server Program. MTA SZTAKI Budapest, 1987.

56. Andó Gy., Gyepesi Gy., Hannák L., Remzsö T., LATOR - a Database Management system for Local Area Networks. Database Definition. MTA SZTAKI Budapest, 1987.

57. Andó Gy., Gyepesi Gy., Hannák L., Remzsö T., LATOR - a Database Management system for Local Area Networks. The MISS Multitask Interface to DOS 3.1. MTA SZTAKI Budapest, 1987.

58. Andó Gy., Gyepesi Gy., Hannák L., Remzsö T., LATOR - a Database Management system for Local Area Networks. The PASCAL Interface. MTA SZTAKI Budapest, 1987.

59. Andó Gy., Gyepesi Gy., Hannák L., Remzsö T., LATOR - a Database Management system for Local Area Networks. The MICROSOFT-C Interface. MTA SZTAKI Budapest, 1987.

60. Andó Gy., Gyepesi Gy., Hannák L., Remzsö T., LATOR - a Database Management system for Local Area Networks. Utilities. MTA SZTAKI Budapest, 1987.

61. Andó Gy., Gyepesi Gy., Hannák L., Remzsö T., LATOR - a Database Management system for Local Area Networks. The LATROQ Guide. MTA SZTAKI Budapest, 1987.

62. Lefons E., Silvestri A., The Use of Multidatabase in Decision Support Systems, In: Distributed Data Sharing Systems (Proc. of the Third International Seminar on Distributed Data Sharing Systems, Parma, Italy) Ed.: Schreiber F.A., Litwin W., North-Holland, 1985., pp 25–43.

63. Manufacturing Automation Protocol / Technical and Office Protocol (MAP/TOP) 1987/1, Ed.: MAP/TOP User's Group of SME, North-Holland, 1987, pp 648.

64. Martin J., Computer Networks and Distributed Processing. - Prentice-Hall, Inc. Englewood Cliffs, N.J. 07632. 1981. - p. 562.

65. Martin J., Design and Strategy for Distributed Data Processing. - Prentice-Hall, Inc. Englewood Cliffs, N.J. 07632. 1981. - p. 624.

66. Microcomputer Decision Support Systems - Design, Implementation and Evaluation, Ed.: Andriole S.J., North-Holland, 1986, pp 358.

67. Musteata B., Lesser R., VSAM Techniques: System Concepts and Programming Procedures, North-Holland, 1986, pp 398.

68. Nickerson R.S., Using Computers - Human Factors in Information Systems, MIT Press, London, England, 1987, pp 456.

69. Remzsö G., Applications software for PC LANs, MTA SZTAKI Tanulmányok,203(1988)., pp 185-195.

70. Remzsö, T., Computerized Data Base Management Systems, Budapest, Információ/Elektronika, 4 /1979, pp 199-2O6.

71. Remzsö T., Data Base Management Systems (in Hungarian), Technical University of Budapest, Doctoral Theses, 1979., p 165.

72. Remzsö T., Computer Aided Information System in the ELEKTROMODUL, 2th International Conference **Neumann János**, Hungary, Székesfehérvár, 1983.

73. Remzsö T., Industrial Computer Aided Information System Based on the Distributed Data Base Management - A Case Study , MTA SZTAKI Tanulmányok, Budapest, 147/1983, pp 169-174.

74. Remzsö T., Office Automation and data processing. - MTA SZTAKI Tanulmányok, Budapest,194(1986), pp 191-201.

75. Remzsö T., Computer aided management system in the Hernád "Március 15." Agricultural Co-operative (A Case Study). - MTA SZTAKI Tanulmányok, Budapest, 194(1986), pp 183-190.

76. Sedgewick R., Algorithms, Addison-Wesley Publishing Group, Amsterdam, 1988, pp 640.

77. Simons G.L., Management Guide to Office Automation, John Wiley & Sons Ltd, England, 1986, pp 370.

78. Stuck B.W., Arthurs E., A Computer & Communications Network Performance Analysis Primer, Prentice-Hall, Inc., Englewood Cliffs, New Yersey, 1985, pp 532-588.

79. Tangney B., O'Mahoney D., Local Area Networks and Their Applications, Prentice Hall International, UK, 1988, pp 240.

80. Van Amstel J.J., Poirters J.A.A.M., The Design of Data Structures and Algorithms, Addison-Wesley Publishing Group, Amsterdam, 1988, pp 320.

81. Wilkinson G., Winterflood A., Fundamentals of Information Technology, John Wiley & Sons Ltd, 1987, pp 382.

Приложение 1.

Список синтаксических ошибок

Номер кода синтектических ошибок появляется в поле **DB-SYNTAX-ERROR** (в буффере повторения). Имя считается неизвестным в команде, если оно не определено в логической схеме.

1 - Имя файла неизвестно.
2 - Ключ неизвестен.
3 - Связь неизвестна.
4 - Имя поля или неизвестно или не принадлежит записям файла.
5 - Поле ключа неизвестно.
6 - Неправильная дата. Или год меньше чем 1900 или дата длиннее чем 20760131.
7 - Несуществующий месяц.
8 - Несуществующий день.
9 - Отсутствует разделители команд.
10 - Отсутствует имена разделителей.
11 - Отсутствует разделитель даты.
12 - Слишком длинная команда.
13 - Пользователей больше, чем зарегистрировано.
14 - «сторона» в FATT не LEFT и не RIGHT.
15 - Связь не имеет текущей линии.
16 - Неправильное имя файла в координате связи.
17 - Преждевременное окончание команды в CORR, TERR и DIR.
18 - Команда не может быть интерпретирована.

19 - Несуществующая команда.

20 - Команда от нераспознанного пользователя.

21 - Статус события - "нет" в команде, где необходима текущая запись или линия.

22 - Семафор не существует.

23 - «связь » в команде не симетрична.

24 - Фатальные ошибки: переполнен диск, не читается системной журнал, нет места для системного журнала.

Приложение 2.

Параметры и сообщения об ошибках **SERVER**

Как уже упоминалось, для функционировании **LATOR**-у необходимо последовательно получить значения следующих параметров:

1. Имя параметра файла.
2. Имя БД.
3. Стартовый режим (0 - обычный, 1 - обновление из командного файла).
4. Режим окружения (0 - тестовый, 1 - обычный.)
5. Размер страницы.
6. Номер страницы.
7. Номер страницы для индексных файлов (VSAM).
8. Наибольшее число станций.
9. Область пользователя (число даётся в байтах, наибольшее 480).
10. Системный журнал (0 -нет, 1 - да.)
11. Размер области расширения для системного журнала (количество блоков по 512 байт.)
12. Командный файл (0 -нет, 1 - да.)
13. Цикл закрытия файлов и рестарта системного журнала (наибольшее число открытых транзакций.)
14. Наибольшее число "запираний".
15. Наибольшее число "семафоров".
16. Автоматическое запирание (0 - нет, 1 - да.
17. Эхо.

LATOR выдаёт также сообщения об ошибках:

Недостаточно памяти.
Схема и БД несоответствуют.
Не открыта логическая схема.
Не открыт каталог.
Логическая и физическая схемы не соответствуют.
Преждевременное окончание в физической схеме.
Физическая схема и каталог не соответствуют.
Слишком длинная линия в физической схеме.
Неизвестная часть в командном файле.

Первой активизирована **MISS**.
Ошибка в каталоге.

MISS не в памяти или **LATOR** уже выполняется.
Системный журнал и схема не соответствуют.
Не открыт символьный файл.
Ошибка в символьном файле.

Нестранимые ошибки **LATOR** следующие:

Синтактическая ошибка из станции «№ станции».
Нет памяти для системного журнала.
Системный журнал не может быть прочитан.
Переполнен диск.

Приложение 3.

Структура локальных распределённых систем

Возрастение объёма передаваемой и обрабатываемой информации на различных объектах требует создания высокопроизводительных структур, которые реализуются в локальных распределённых системах (ЛРС). В основе таких систем лежат принципы параллельного выполнения большого числа операций, переменной логической структуры, конструктивный унификации. Для организации взаимодействия элементов таких систем используются локальные сети передачи данных (ЛС ПД).

В локальных распределённых системах вычислительные ресурсы систем обработки информации (СОИ), а также компоненты ЛС ПД размещаются в непосредственной близости от источников информации. Надёжность таких систем обеспечивается за счёт децентрализации обработки данных и управления. В ЛРС реализуется принцип географического распределения оборудования и функций, выполняемых системой. Рассматриваемые системы в зависимости от их назначения размещаются на территориях диаметром от нескольких десятков метров до километров.

ЛРС, а следовательно ЛС ПД, могут объединяться в территориальные системы с помощью шлюзов или включаться в территориальную сеть передачи данных с помощью интерфейсных станций ИнС.

Вычислительные сети отражают общую тенденцию к распределению информации, проявляющуюся в создании многопроцессорных распределённых систем, баз данных, интеллектуальных терминалов. Уменьшение стоимости процессоров и других компонентов

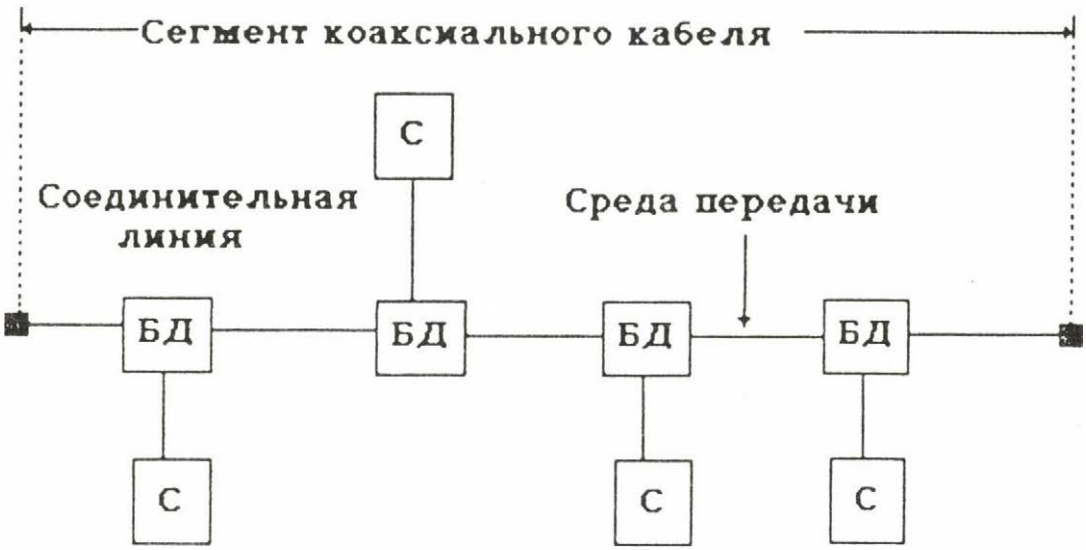
информационных систем устранило необходимость концентрации вычислительных ресурсов и создало предпосылки к применению распределенных систем. Гибкость этих систем такова, что позволяет разделить функции сложной системы не только логически, но и физически.

Локальные распределенные системы поднимают на качественно новую ступень управление производственными объединениями, промышленными предприятиями. Технология локальных сетей (ЛС) оказывает влияние и на структуру вычислительных систем. С её помощью реализуется принципиально новый подход, заключающийся в том, что между пользователями распределяется не время центрального процессора одной ЭВМ, как в централизованных системах, а компоненты базы прикладных процессоров. В соответствии с определённой процедурой доступа пользователю выделяется прикладной процессор. Если мощность процессора оказывается недостаточной, она может наращиватся с помощью дополнительных процессоров.

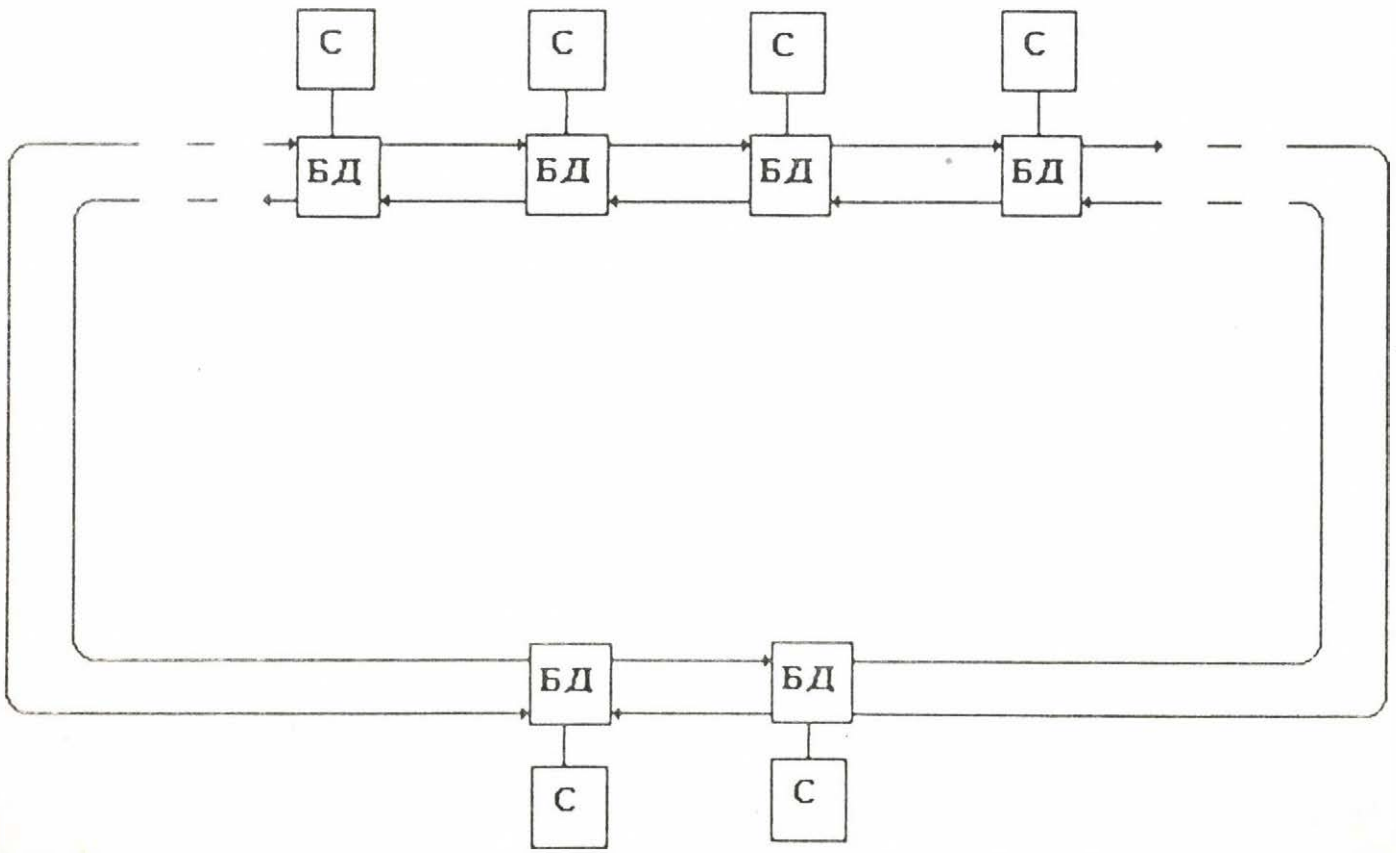
В настоящее время для построения локальных сетей используются два основных вида **топологии** : магистраль (шина) и кольцо.

В случае магистрали все станции параллельно подключаются к одной и той же среде передачи (обычной коаксиальному кабелю или витой паре) и сигналы, передаваемые станцией, поступают ко всем другим станциям, подключённым к среде.

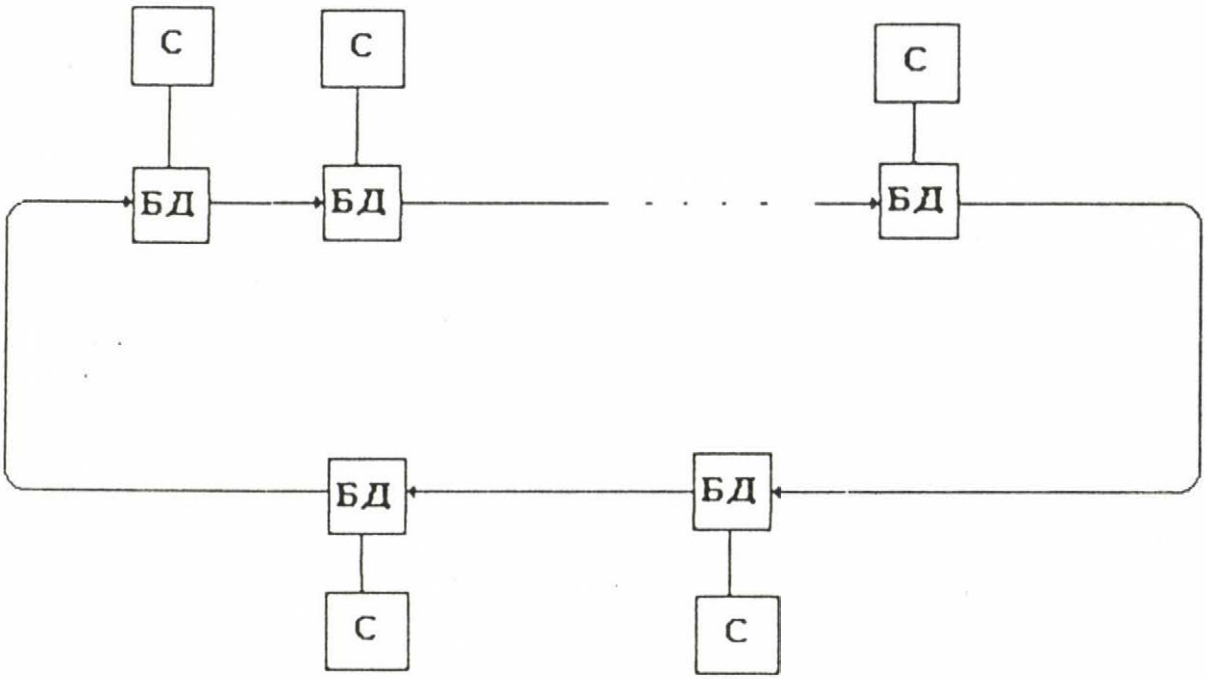
В случае кольцевой топологии передаваемый сигнал обычно поступает к одной (соседней по кольцу) станции, в которой производится ретрансляция принимаемых сигналов к следующей станции и т.д.



Конфигурация сегмента локальной сети с параллельным подключением станций (С - станция, БД - блок доступа к среде передачи)



Конфигурация кольцевой локальной сети
(С- станция, БД - блок доступа)



Конфигурация моноканал с резервированием
(С - станция, БД - блок доступа)

Классификация методов доступа, применяемых в локальных сетях

Все методы доступа, применяемые в локальных сетях, можно подразделить на две категории: методы, базирующиеся на централизованном управлении сетью, и распределённые методы доступа.

Для практического применения в условиях обеспечения высокой надёжности наибольший интерес представляют распределённые методы доступа, в которых центральный управляющий орган отсуствует и все станции сети функционируют автономно. При таких методах доступа сеть более надёжна, поскольку в ней отсуствует критический пункт - центральная станция, отказ которой выводит из строя всю систему. Распределённые методы доступа для локальных сетей в топологией типа шины можно подразделить на четыре основные категории: /28./

- случайные методы доступа
- маркерные методы доступа
- интервальные методы доступа
- интервально-маркерные методы доступа .

A TANULMÁNYOK SOROZATBAN 1988-BAN MEGJELENTEK

203/1988 KNVVT EG-25 Problems and tools of the integration of information systems. Proceedings 1987. Edited by: Rumjana Kirkova, Tibor Remzső, Ferenc Urbánszki

204/1988 Csetverikov Dimitrij: Digitális texturavizsgálat néhány uj módszere

205/1988 Hernádi Ágnes: Uj eszközök a fogalmi modellezésben

206/1988 The second Hungarian workshop on image analysis Edited by: Dimitrij Csetverikov, Géza Álló

207/1988 Suzanne Márkusz - Gábor Márkus: Logic Puzzless and Logic Programming I /Logikai fejtörők - logikai programozás I/

208/1988 Proceedings of the 5th International Meeting of Young Computer Scientists /IMYCS'85/ Edited by: E. Csuhaj-Varju, J. Demetrovics, J. Kelemen

209/1988 Галя Младенова Ангелова: Синтаксические и семантические структуры реляционных языков запросов

210/1988 Publications'1987 - Publikációk'1987 Edited by/Szerkesztette: Petróczy Judit

211/1988 Eszenszki József - Kas Iván - Palotási András Podmaniczky András - Szücs Miklós - Vörös Károly Zalán Frigyes - Alexander Mihajlovics Klocskov Valerij Alexandrovics Plahov:

Tanulmány a számitógépes, raszteres mikrofilm lap készités-elvi és gyakorlati kérdéseiről

212/1988 Густав Хенчей: Модели ассоциативных образов