PREFETCHING IN REALTIME DATABASE APPLICATIONS

H Wedekind and G. Zoerntlein

Universitaet Erlangen-Nuernberg
Informatik VI
Martensstrasse 3, 8520 Erlangen
W-Germany

Abstract·

In this paper a method is proposed how to achieve response times of main memory database systems without keeping the whole database in main memory The method was originally developed for realtime systems in manufacturing automation, but it is applicable in environments where canned transactions interact with databases rather than people performing free transactions The main idea is to preanalyse canned transactions in order to extract knowledge about their local access behaviour. This knowledge is used by the runtime system of the database when the transaction is started Concepts for modules doing the preanalysis and the runtime tasks are described in detail Furthermore a database architecture is developed incorporating these new components.

Key words

Prefetching, main memory database, transaction schema, canned transactions, preanalysis

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery To copy otherwise, or to republish, requires a fee and/or specific permission

© 1986 ACM 0-89791-191-1/86/0500/0215 $00 75

1 Description of the application environment

1 1 Components of realtime systems

Typical realtime applications arise from the field of production automation There are a lot of tasks where the computer and the machine control must respond to certain inputs within a given span of time, otherwise the entire system is running out of control such tasks are called realtime applications The following outline describes the main components of a realtime application system

Fig 1 Schematic representation of realtime system components

A realtime application system consists of·

(1) the technical system
it consists of all the machines performing physical activities The machines may be regarded as finite state automata. The states of the machines are building up the state of the technical system

(2) the steering system
it comprises all the components which may cause a machine to change its state and therefore influence the state of the technical system

(3) the sensor and alarm system
it covers all instruments providing information about the present state of the technical system

(4) the process computer
it receives data from the measuring and alarm system and directs its output to the steering system In this paper the entire software of the process computer accepting data from the measuring and alarm system and sending signals to the steering system will be called a realtime system. Alarms and measuring data together are named external events

Realtime systems work as follows:

At a certain time t_1 not being predictable, a certain external event e arises and is transmitted to the process computer Such events are for example the triggering of an alarm signal or the interruption of a light barrier caused by an arriving work piece Because of the event and the captured information i_e, the realtime system has to perform certain actions $a_1, \dots, a_n$ causing the system to supply a response r to the event e It is essential that the response r is provided within a fixed interval Δt after occurrance of event e It is the task of the realtime system to combine the captured information i_e with known information i_{pre} and to compute the actions necessary to initiate the reaction r. The information i_{pre} is composed of information provided by previous events and parameters of the technical system and must be managed by the process computer

Realtime system are getting more complex This means in particular that information i_{pre} used in computing the response r to an event e, becomes more extensive The growing volume of data and the more intricate data structures require a better organization of data management in realtime systems
The organization of a large and complex amount of data was profoundly studied in commercial data processing resulting in application independent database management systems (e g SYSTEM R, ADABAS, INGRES etc) These commercial database management systems haven't been used by realtime systems up to now It is argued that these database management systems are too slow because of their universal conception and that there is no guarantee of system's response within a prefixed interval In this paper a concept is developed supporting the minimization of response times of database management systems for a specific class of applications The development was governed by the following two assumptions
(1) the whole database buffer is in main memory and
(2) the main memory is limited to a rather low size
The first assumption may be due to the fact that either the operating system doesn't support any virtual memory management or that there are utilities allowing to fix a certain realm in the main memory The second assumption doesn't allow to keep the entire database in the database buffer This is often discussed in the environment of mainframe computers where up to 10 MByte of buffer are provided These assumptions rather meet the constraints we were facing in our project on manufacturing automation using process computers

1 2 Transactions in realtime systems

An important concept in database applications is the notion of transactions A transaction is a sequence of elementary database operations $\langle o_1, \dots, o_n \rangle$ performing as a whole a consistent database transformation. Starting with a consistent state of the database a new consistent state is achieved after execution A very important classification for the ideas developed in this paper is the distinction between canned transactions and free transactions /ST 84/

Canned transactions are characterized as transactions where the type and the sequence of the elementary database operations are determined at the beginning of the transaction (BOT) The elementary database operations contain parameters and the only task is to substitute them by actual values when the transaction is performed Usually it is known whether such an elementary database operation is a read, write or update operation, it is known likewise what bigger parts (e g segments, relations, parts of relations, attributes) of the database are affected The primary notion in the course of database system's development was the fixed framework of a canned transaction Many actions not only in business are done in a highly stereotype, unalterable and repetitive way, just for the indispensible sake of efficiency and productivity
Within free transactions number and type of various database operations are unknown when the transaction is started This kind of transaction is typical for office automation where skilled workers interact as problem solvers with the database system in an unregulated way In production automation, however, it is attempted to steer and control technical systems by programs

For understanding the concepts developed in this paper the term transaction schema is helpful To avoid misunderstandings a brief outline of a transaction schema is given now

Canned transactions are usually embedded in application programs especially in the environment of realtime systems These programs have a control structure (sequences, alternative paths, parallel activities

etc.). Database operations parameterized by valuechanging program variables are incorporated into the control structure The control structure of a program apart from the concrete program statements is called the program schema Now a transaction schema is defined as a program schema with integrated parameterized database operations starting with a BOT-command (BOT = begin of transaction) and ending with an EOT-command (EOT = end of transaction).

A great formalism is needed to define the notions "program schema" and "transaction schema" precisely Instead of presenting such a formalism contributing not very much to the fundamentals of our concept the following graphical elements are used to represent program and transaction schemata

(1) if — represents alternative program paths,

(2) while — represents a program part that can be active several times;

(3) fork — leads into program parts being active in parallel,

(4) join — denotes the joining of program activities running parallel,

(5) BOT — denotes the beginning of a transaction,

(6) EOT — denotes the end of a transaction,

Figure 2 shows an example of a typical realtime transaction schema Besides the graphical elements explained above notations are introduced to indicate "waiting for an event" (e_1, ,e_6), "initiating a reaction" (r_1, ,r_6) and "performing a database operation" (o_1, ,o_6) resp The first two types of statements are very important for realtime applications.

The functionality of the realtime transaction given in figure 2 may be described in the following way

(1) After the start of the transaction (BOT) the program waits for the external event e_1 After e_1 occured the program executes the database operation o_1 and computes the reaction r_1

(2) Then two activities will run in parallel, indicated by a fork statement One activity is waiting for the event e_2 to perform the necessary operation o_2 and to initiate the reaction r_2 after the occurrence of e_2 Meanwhile the other activity looks for event e_4

(3) Having performed their tasks the two activities are synchronized, indicated by a join statement

(4) Depending on a certain condition the transaction follows one of the two program branches, either it is controlling event e_5 and performing operation o_5 to compute reaction r_5 or it is waiting for event e_6

A particular occurrence of a transaction schema is a linear sequence of database operations These database operations have been derived from the operations in the transaction schema by instantiating the parameters The operation sequence is ruled by the program schema and is established when transferring the control to the database system Figure 3 shows a possible occurrence of a transaction according to the transaction schema described in figure 2

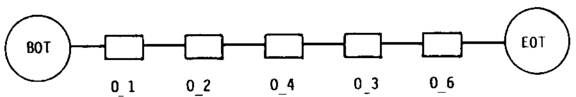


Fig 3 Possible transaction according to schema in figure 2

If there is no confusion the term transaction will also be used to denote a transaction schema as well as the occurrence of a transaction schema

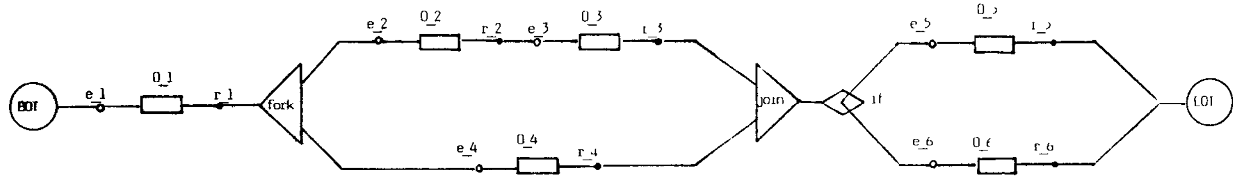
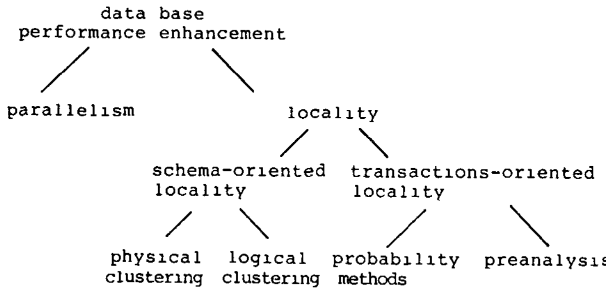


Fig 2 An example of a typical realtime transaction schema

1.3 Access optimization



Mainly, there are two ways to enhance performance of database systems. One way is to exploit parallelism, the other to improve local access behaviour. Local access behaviour - in short locality - may be introduced in different ways. We distinguish two kinds of locality:

- schema oriented locality, an optimization method during schema design, that comprises clustering, file inversion and link, e.g. link of database entities being known to accessed in sequence.
- transaction-oriented locality that uses information about the access behaviour of transactions. Information about transaction access behaviour may be gained in two different ways, either using probability methods or transaction preanalysis.

We will emphasize transaction preanalysis that seems to be promising in our application environment. The concept proposed in this paper may be explained as follows. By analysing a program containing a particular transaction, potential sequences of database operations are extracted. This means in mathematical terms that an ordering is established on the set of database operations being described by the transaction schema. The knowledge about this ordering is kept in the database and will be used when the program is started. This concept will be elaborated in the following chapters.

2. The concept of prefetching

2.1 Preanalysis of transactions

According to our distinction realtime application of database systems in production automation are classified as canned transactions embedded in programs. There are two ways to incorporate database operations into a program. The first type is the direct integration, the second one is the call interface. Direct integration can be achieved either by precompilation or by extending the compiler of the host language. The call technique treats database operations as external procedure calls and usually the compiler doesn't distinguish procedures concerning databases from any other external procedure. For our purposes presented in this paper the degree of integration is of no concern. It is only required that there is a module that analyses the realtime application program before its first execution. This module must be able to recognize

- DML-statements,
- statements that spawn concurrent processes such as the fork/join construct or the cobegin/coend statement,
- statements that define program branches like the wellknown if-then-else statement, the case statement but also statements that define loops like the while statement or the repeat statement.

The module may be a precompiler, a part of the host language compiler or a special program developed to implement the presented concept.

Since precompiling is a very elegant way to embed database operations into programs our concept will be explained with regard to this integration technique. Using precompilation the programmer must formulate the database operations in a database manipulation language and mark them with special characters (INGRES uses ##) in order to separate them from other program statements. In a special compilation phase preceding the normal program compilation the database operations are transformed into a syntax accepted by the compiler of the host programming language. The database operations o_i are transformed into external procedure calls having the general form $P_i(s_{i,1}; \ldots ,s_{i,n})$. The external procedures are called access modules. When executing the transaction the parameters $s_{i,1}; \ldots ;s_{i,n}$ are replaced by actual values. The parameters of such a procedure P_i may be separated into two categories. The first includes those parameters whose values are already known before the event e_i enters (e.g. a relation name, an attribute name or the value of an attribute). The second is formed by the parameters receiving their values from event e_i.

The main task of such a procedure P_i is to qualify a set $E(P_i)$ of pages or blocks on disk storage. Now it is possible to assign to each procedure P_i a procedure $P_i`$ with the following properties:

(1) $P_i`$ only includes parameters whose values are already known before the event e_i.

(2) $P_i`$ qualifies a set $E(P_i`)$ of disk pages comprising the set $E(P_i)$, i.e. $E(P_i`)$ is a superset of $E(P_i)$.

(3) $P_i`$ doesn't make any changes in the qualified disk pages.

The meaning of these properties will be shown in the following section

Figure 4 summarizes our notions of preanalysing transactions with regard to the example given in figure 2

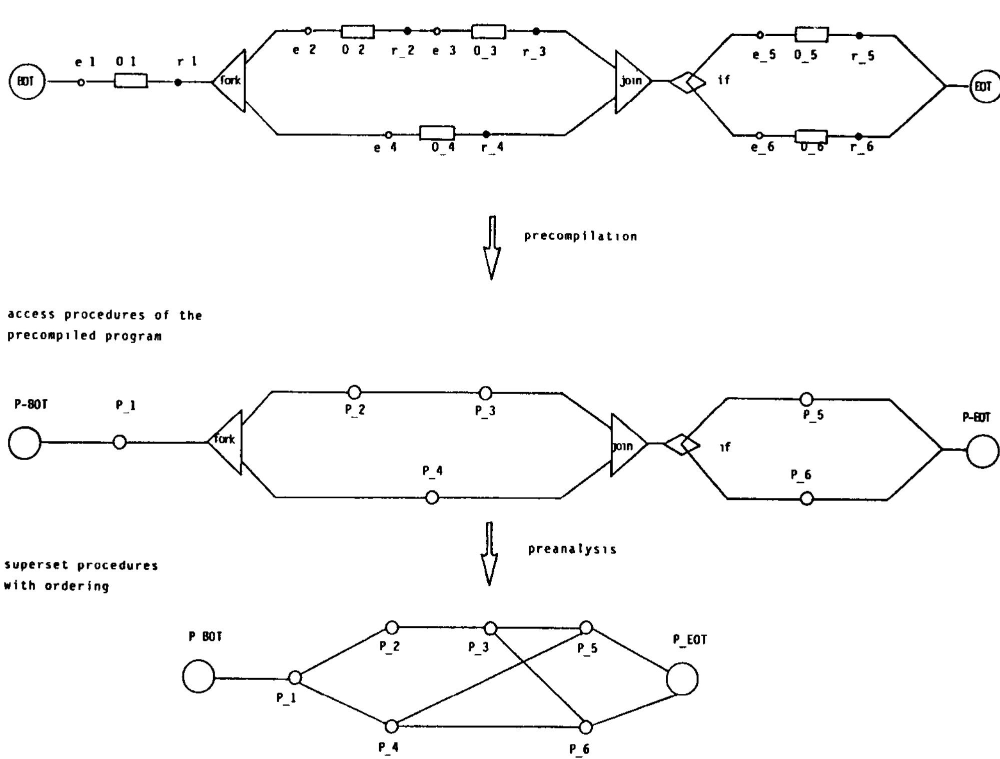


Fig. 4. Precompilation and preanalysis of a canned transaction

2.2 Transaction oriented prepaging in database management systems

The transfer of a disk page into the main memory takes up 1000 machine cycles Thus it is desirable to hold the pages needed by procedure P_i in the main memory when P_i starts its execution This can be accomplished by proceeding as follows

When preanalysing the realtime program in which the transaction is embedded a list is produced including the procedures P_i` with properties stated in 2 1 for each procedure P_i. Furthermore an ordering of this set of procedures is described being derived from program flowcharts and representing the possible sequences of the database operations To derive the precedence structure the analysis module has to know the language constructs indicating parallel activities and the statements leading into alternative program branches

Each list receives a transaction identification This identification together with an identifier distinguishing the different DML-statements is appended to the procedures P_i with the unknown parameters The list of procedures being constructed together with the precedence structure and the identifications are transmitted to the database management system to handle them When a realtime program starts a transaction T by issuing a BOT-command the database management system recognizes the activated transaction via the attached identification Immediately after the BOT-command is received the database management system executes all the procedures P` at the top of the precedence structure and fixes the qualified disk pages in the database buffer in order to prevent other pages from replacing them. After an event e_i has entered, the realtime program initiates the execution of procedure P_i When procedure P_i is evaluated no data from the disk are needed since the necessary pages are still in the database buffer The execution time of procedure P_i will be reduced essentially After the evaluation of procedure P_i the fixed pages are released and all the

successors of P_i` are started by the database management system Further steps are obvious now Proceding in this way means that the execution time for operation o_i is minimal from a global point of view We do not consider local minimums achieved by particular control mechanisms, e.g priority control.

Summarizing the discussion our concept may be characterized in the following way Before executing realtime programs containing database transactions a second transaction is created with database operations derived from the operations of the realtime transaction by applying conditions stated in 2.1 This second transaction is started together with the realtime transaction, and is synchronized with it via the database operations using a special module of the database management system.

The concept of prefetching is covered by the more general concept of locality, an optimization approach in database systems, which is as important as the techniques of parallelism The pages being attached or being fixed to particular transactions are local to them, and thereby fetched without considering the "global space" of databases Prefetching is a technique to construct a dynamic, transaction-oriented mainmemory sub-database.

3 Implementation concepts

3.1 Architecture of a realtime database management system

To realize the ideas described in the previous sections a database architecture slightly different to conventional ones is proposed The new components of this architecture and their interaction with traditional parts of a database management system are shown in figure 5

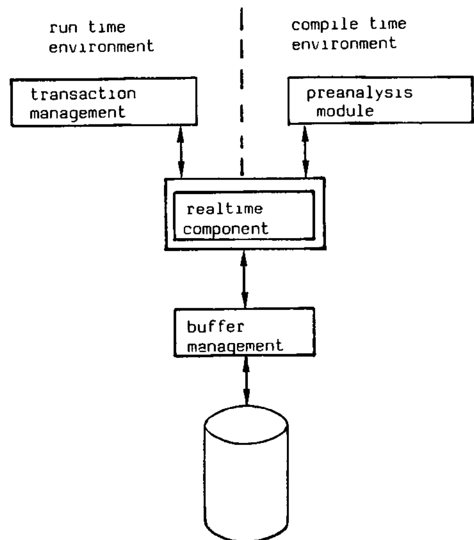


Fig 5 Architecture of a realtime database management system

A central part of this architecture is a realtime component, an extension of the kernel of the database management system This component will interact with the preanalysis module when a realtime transaction is being preanalysed and it interacts with the transaction management during the execution of a preanalysed transaction. In section 3 2 we discuss how to construct the superset procedures, i.e. the compile time environment is described, and in section 3 3 we consider in some detail the runtime environment.

So far the presented ideas are independent of any data model The following implementation concepts, however, have to refer to a specific data model. Because of its clarity and its simplicity the relational data model is chosen. It is assumed that the reader is familiar with it The outline in section 3.2 is mainly given for notation purposes

3.2 A concept for the implementation of the preanalysis module

The proposed method produces the procedures P_i` by transforming the original database operations into those known in advance and compiling these modified operations. The method is described in detail now referring to the relational data model.

The notion of a relation schema is very important for understanding the relational data model Such a relation scheme has the form $R(A_1; \ldots, A_m)$ where R stands for the relation name and $A_1, \ldots, A_m$ are called the attributes of R Each attribute A_i has a domain denoted by $dom(A_i)$. At any time t the stored data in the database define a relation R_t with the property:

$$R_t \subseteq dom(A_1) \times \ldots \times dom(A_m)$$

The elements of R_t are called tuples

Following /DB 82/ a database schema $\mathcal{S}$ may be defined as a triple (S,O,C) with:

(1) $S = \{R_i(A_{i,1}; \ldots; A_{i,mi}) \mid i = 1, \ldots; n\}$ a set of relation schemes;
(2) O a set of relational algebra operations applicable to S;
(3) C a set of semantic integrity constraints

To denote operations out of O a subset of the data manipulation language QUEL is used. This language is derived from the relational calculus. It has a precise syntax and is easy to understand. It uses the term of a tuple variable t which denotes a variable assigned to a relational schema R. The values of t are tuples out of R_t. An

indexed tuple variable is a term t A, where t is a tuple variable and A is an attribute of range of R The term t A denotes a variable having values out of dom(R A)

The elements in O are of the following types

- APPEND R (<target_list>) WHERE <qualification>,
- DELETE t WHERE <qualification>,
- REPLACE t (<target_list>) WHERE <qualification>;
- RETRIEVE t (<target_list>) WHERE <qualification>

To produce a procedure P_i` with respect to a given database operation op of the O it is proposed to transform op into a database operation op` with the following properties·

(1) op` is of the type RETRIEVE
(2) op` qualifies a set of tuples comprising the set of tuples specified by op; i e op qualifies a superset of op

It will be shown now how property (2) can be achieved

A qualification <qual> is a boolean combination of clauses of the type.

$$cl = \text{<term_1>} \ \theta \ \text{<term_2>}$$

where $\theta \in \{=,\neq,<=,<,>=,>\}$ and where <term_i> i = 1,2 has one of the following forms

- <term_i> is a constant value,
- <term_i> is an indexed tuple variable t A_i,j;
- <term_i> is a program variable v

Considering tuple qualifications the clauses must consist at least of one term being a tuple variable So without loss of generality it may be assumed that <term_1> is always a tuple variable Each qualification can be transformed into the disjunctive normal form (DNF) So it may be written:

$$\text{<qualification>} = \bigwedge_{1 \leq r \leq p} \bigvee_{1 \leq s \leq qr} (cl_r,s) \ ,$$

where $\bigwedge$ denotes conjunctions and $\bigvee$ denotes disjunctions

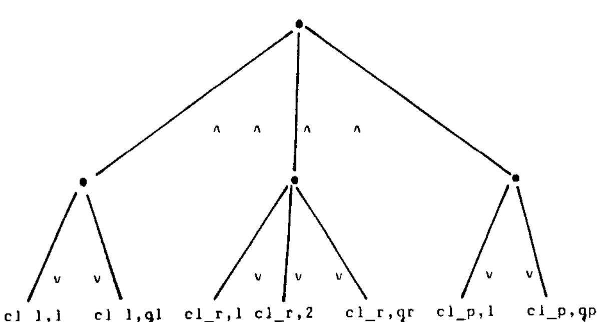


Fig. 6: Qualifications in disjunctive normal form

Let us consider a realtime transaction containing a database operation op_i with the qualification <qual_i> Two types of clauses may be distinguished in <qual_i>

(CL1) clauses where the value of <term_2> is fixed before event e_i;
(CL2) clauses where the value of <term_2> is determined by event e_i

The qualification <qual_i`> is gained by applying to <qual_i> the following operations

1. Remove from <qual_i> all clauses of type (CL2).

2. Compare the new clauses with the old one. If there is a tuple variable t in the old clause that doesn't appear in the new one then add a disjunction of the type t A = * to the new clause, where A is a primary key of the relation the variable t is referring to

The evaluation of expressions of type t.A=* causes the qualification of all the tuples of the relation the variable t is referring to

The tuples qualified by <qual_i> are a subset of the tuples qualified by <qual_i`> Removing a conjunction means that the qualified tuple set is enlarged. The conditional insertion of clauses of type t A = * ensures that no relation can disappear

The following schema comprises our notions of how to gain the superset procedures P_i`

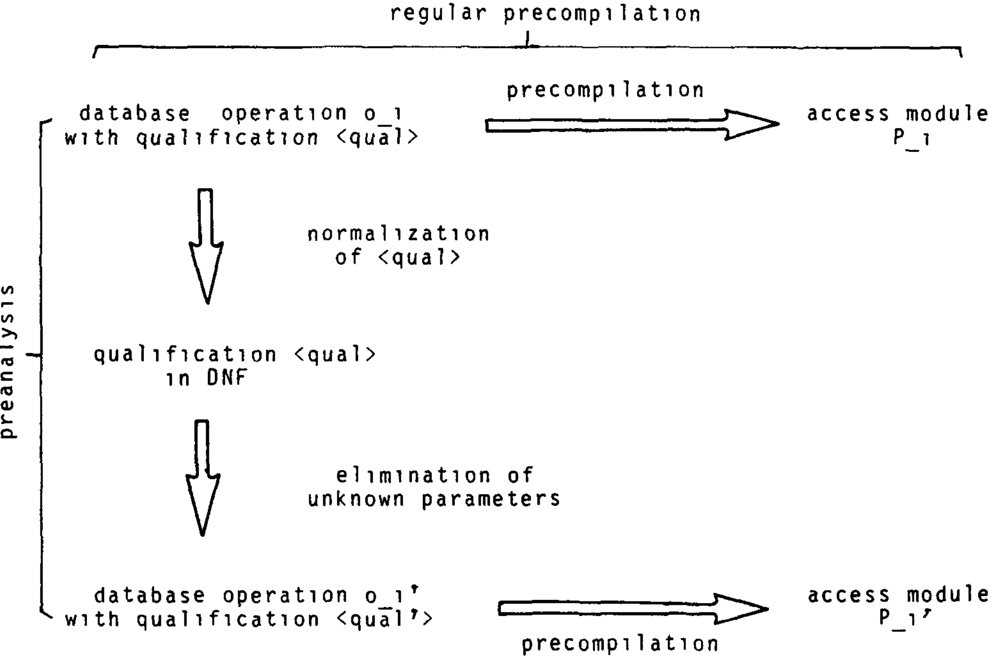


Fig. 7 Derivation of the superset procedures

3.3 The runtime environment

In our database architecture (see figure 5) the interaction of the realtime component with the transaction manager is explained in more detail now. The main data structures of the realtime component are described first. Then in 3.3.2 the synchronization of a realtime transaction in execution and its derived superset transaction is analysed in some detail. Finally in 3.3.3 the functions are specified that must be provided by the realtime component.

3.3.1 The data structures

The realtime component contains information about all preanalysed transactions as well as the current states of realtime transactions being active. The main data structures and the main functions of such a component are described in the following subsections.
For the description of the data structures the relational data model is used. Attributes being underlined denote primary keys. There are only three main data structures.

a) The qualification list

The superset procedures P_i' of all preanalysed realtime transactions are stored in this list. The data structure can be represented in the following way.

O-QUAL (<u>proc-id</u>, proc-code)

b) The ordering list

The information about the precedence structure on the data base operations of the preanalysed realtime transactions is kept in the ordering list. It is composed of the following attributes:

PRE-STRUCT (<u>predecessor, successor, trans-id</u>)

For the attributes the following constraints must hold:

- dom (PRE-STRUCT.predecessor) = dom (O-QUAL.proc-id)
- dom (PRE-STRUCT.successor) = dom (O-QUAL.proc-id)

c) The active list

As soon as a preanalysed realtime transaction is activated new elements are added to the active list. These elements show the current states of the active transactions. The list is updated when a database operation has been executed by a realtime transaction. The relation is of the following form.

ACT-TRANS (<u>active-id, current-state, trans-id</u>)

For attributes of this relation the following conditions must hold.

o dom (ACT-TRANS.current-state) = dom (O-QUAL proc-id)

o dom (ACT-TRANS.trans-id) = dom (PRE-STRUCT.trans-id)

In addition the following semantic integrity constraints must be fulfilled in our database·

$$(1) \bigvee o \; (o \in \text{O-QUAL} \wedge o.\text{proc-id} = \text{P_BOT} \wedge o.\text{proc-code} = \text{dummy})$$

$$(2) \bigvee o \; (o \in \text{O-QUAL} \wedge o\ \text{proc-id} = \text{P_EOT} \wedge o.\text{proc-code} = \text{dummy})$$

$$(3) \bigwedge p \; (p \in \text{PRE-STRUCT} \dashrightarrow \bigvee o \; (o \in \text{O-QUAL} \wedge o.\text{proc-id} = p.\text{predecessor}))$$

$$(4) \bigwedge p \; (p \in \text{PRE-STRUCT} \dashrightarrow \bigvee o \; (o \in \text{O-QUAL} \wedge o\ \text{proc-id} = p.\text{successor}))$$

$$(5) \bigwedge p \; (p \in \text{PRE-STRUCT} \dashrightarrow \bigvee q \; (q \in \text{PRE-STRUCT} \wedge q.\text{trans-id} = p\ \text{trans-id} \wedge q.\text{predecessor} = \text{P_BOT}))$$

$$(6) \bigwedge p \; (p \in \text{PRE-STRUCT} \dashrightarrow \bigvee q \; (q \in \text{PRE-STRUCT} \wedge q\ \text{trans-id} = p\ \text{trans-id} \wedge q.\text{successor} = \text{P_EOT}))$$

$$(7) \bigwedge a \; (a \in \text{ACT-TRANS} \dashrightarrow \bigvee o \; (o \in \text{O-QUAL} \wedge a.\text{current-state} = o.\text{proc-id}))$$

$$(8) \bigwedge a \; (a \in \text{ACT-TRANS} \dashrightarrow \bigvee p \; (p \in \text{PRE-STRUCT} \wedge a.\text{trans-id} = p.\text{trans-id}))$$

For the example given in 1 2 (figure 2) the relations O-QUAL and PRE-STRUCT would contain the tuples:

O-QUAL

proc-id	proc-code
P_BOT	dummy
P_EOT	dummy
P_1	P_1`-code
P_2	P_2`-code
P_3	P_3`-code
P_4	P_4`-code
P_5	P_5`-code
P_6	P_6`-code

PRE-STRUCT

predecessor	successor	trans-id
P_BOT	P_1	T-EXAM
P_1	P_2	T-EXAM
P_1	P_4	T-EXAM
P_2	P_3	T-EXAM
P_3	P_5	T-EXAM
P_3	P_6	T-EXAM
P_4	P_5	T-EXAM
P_4	P_6	T-EXAM
P_5	P_EOT	T-EXAM
P_6	P_EOT	T-EXAM

Fig 8· The relations for the example real-time transaction

3 3.2 The synchronization of a realtime transaction and its derived superset transaction during runtime

It was already stated that our concept to improve response time of database operations embedded in a canned transaction is to run an appropriate second transaction in parallel The second transaction is derived from the original transaction during a pre-analysis phase Its execution is under control of the runtime component integrated in our suggested database architecture (see figure 5 in 3.1) The synchronization of the two transactions performed by an interaction of the transaction manager and the realtime component will be explained now using petri nets. The petri nets (figure 10 and 11) refer to the general form of a transaction schema shown in figure 9 The notations have the same meaning as in figure 2

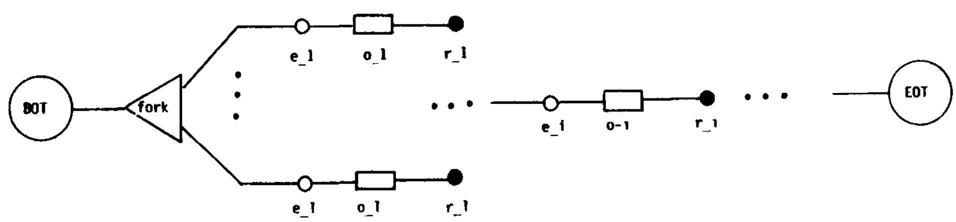


Fig. 9· General form of a realtime transaction

Furthermore it is taken into consideration that several activations of one and the same realtime transaction schema exist simultaneously. Therefore a second index is provided denoting to the activation identifier. It is maintained in the active list of the realtime component Two cases have to be distinguished the start of a transaction and the processing of an event These two cases are shown in the following figures

1 Starting a preanalysed transaction:

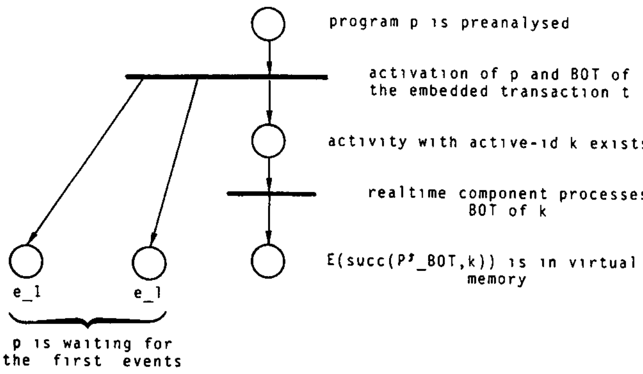


Fig. 10: Petri net describing the start of a transaction

2. Processing an event e_i

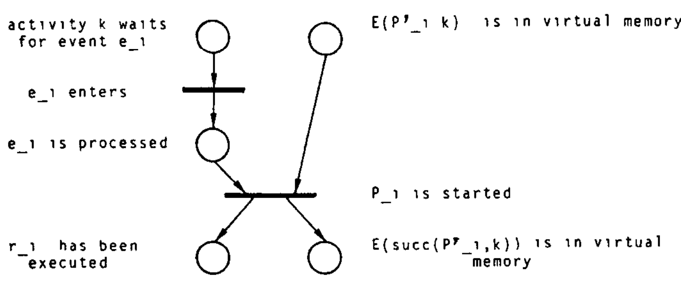


Fig 11 Petri net describing the processing of an event

3 3 3 The functions of the realtime component

The functions being provided by the realtime component of our system may be divided into two groups. The first group is composed of the functions used at compile time, the second one contains the functions used by the runtime environment. These functions are described now using the syntactical conventions of MODULA-2 DEFINITION MODULE declaration. MODULA-2 is a Pascal-like programming language In the following outline attributes of relations are used as TYPE-identifiers:

```
DEFINITION MODULE compile-environment;

  EXPORT QUALIFIED
    trans-id, proc-code, proc-id,
    get-new-trans-id, insert-qual,
    insert-pre-struct,

  TYPE trans-id, proc-code, proc-id;

  PROCEDURE get-new-trans-id () : trans-id;
  PROCFDURE insert-qual
                (c . proc-code) :  proc-id;
  PROCEDURE insert-pre-struct
                (t trans-id, p,s · proc-id);

END compile-environment
```

```
MODULE runtime-environment;

  EXPORT QUALIFIED
    trans-id, proc-id, active-id,
    init-trans, step-trans;

  TYPE trans-id, proc-id;

  PROCEDURE init-trans
                (t · trans-id)   active-id;
  PROCEDURE step-trans
                (t : trans-id, c : proc-id);

END runtime-environment
```

The semantics of the functions introduced above is self-explaining and a more formal description wouldn't contribute much to a better understanding of the concepts.

4. Future Research

The concept presented so far may be improved by the following measures:

a) Future oriented page replacement

The main task of the buffer management is to realize a replacement strategy. Such a replacement strategy selects pages in the buffer to provide storage for a requested page, if a logical reference to the buffer fails. The algorithms applied so far to do this work in database systems only use information of the past for their selection decision (e.g. the age of a page since the first reference, or the most recent reference to the pages etc. /EH 84/) However, empirical studies compared with theoretical work in this field show that replacing algorithms using information of references in the future would produce better database performance In particular Belady /BE 66/ showed that an overall optimal replacement can be attained if a complete knowledge of future references is available Our concept collects information about future page references of transactions during the preanalysing phase and processes this information when this transaction is active. Providing appropriate functions in the realtime component the information about the references of active transactions could also be used by the buffer management to produce a more efficient buffer allocation. One should try by means of new buffer replacement algorithms to approximate Belady's optimum.

b) Minimizing the supersets

One of the most important tasks in our concept is to produce procedures qualifying minimal supersets. This can be reached by capturing the values of as much procedure parameters as possible In the concept presented so far, however, only those parameters are exploited with values being already known at compile time. Using the terminology of chapter 1 a set of parameters in the realtime database operation o_i may be classified as parameters not known at compile time but already known before the event e_i To exploit also these parameters for the superset procedure P_i' a conversation mechanism between active transactions and the realtime component could be developed, where parameter values are passed from the active transaction to the realtime component.

c) Concurrency feature

Our concept tells nothing about concurrently active transactions and a method of synchronizing them. Ordinary locking mechanisms are used in database management systems to guarantee serial schedules of concurrent transactions. Locking, however, may produce states where a transaction can't procede because a requested data object is locked. Such states are unacceptable in realtime systems, because fixed response times can't be guaranteed any longer Since it is known which data objects are used by preanalysed realtime transactions, it may be determined whether interference problems arise with regard to already active transactions. If a transaction is facing unsurmountable problems, e g an untolerable waiting line in front of locked data objects, then provisions should be made to suppress the start of the transactions and to initiate appropriate actions in the technical system

d) Dynamic switch of paging modes

If the supersets get too large and if the system tends to become I/O-bounded prefetching may not pay off anymore because many untouched pages are then fetched. So the technique becomes countereffective. This will certainly be the case if the I/O system tends to be fully loaded over a long time. The question arises whether such hardware systems disqualify a priory for realtime applications. It is out of the scope of this paper to set up a model determining the critical operating point from which an conventional demand paging is preferable A system combining both strategies switching from prefetching to demand paging and vice versa is conceivable.

Literature

/BA 84/ Baumann,R.
Datenverarbeitung unter Zeitbedingungen.
in Informatik Spektrum, Band 7, Heft 2, April 1984

/BE 66/ Belady,L A
A study of replacement algorithms for virtual storage computers
IBM Syst. J 5,2 (1966), 78 - 101

/DB 82/ Dayal,U.; Bernstein,P.A
On the correct translation of update operation on relational views
ACM TODS, Vol 8, No 3, Sep 1982

/EH 84/ Effelsberg,W.; Haerder,T.
Principles of database buffer management.
ACM TODS, Vol.9, No 4, Dec 1984

/GA 83/ Garcia-Molina,H.
Using semantic knowledge for transaction processing in a distributed database
ACM TODS, Vol. 8, No.3,June 1983

/HA 78/ Haerder,T
Implementierung von Datenbanksystemen
Hanser-Verlag, Muenchen 1978

/RO 76/ Rodriguez-Rosell,J
Empirical data reference behaviour in database systems
in Computer, Vol 9, No 11, (1976), pp 9-13

/SM 78/ Smith, A.J
Sequentiality and prefetching in database systems.
ACM TODS, Vol 3, No 3, (1978) pp 223-247

/ST 84/ Steinbauer,D
Transaktionen als Grundlage zur Strukturierung und Integritaetssicherung in Datenbank-Anwendungssystemen
Arbeitsberichte des IMMD d Uni Erlangen, 16 (1984)

/WH 76/ Wedekind,H , Haerder,T
Datenbanksysteme II
B I , Reihe Informatik, Band 18, Mannheim 1976