STEPWISE SPECIFICATION OF DYNAMIC DATABASE BEHAVIOUR

Udo W. Lipeck

Institute fur Informatik
Technische Universitat Braunschweig
Postfach 3329, D-3300 Braunschweig
West Germany

ABSTRACT: This paper presents a methodology for the stepwise specification of dynamic database behaviour. A conceptual schema is described in three levels: data, objects and transactions. To determine which sequences of database states are "admissible", integrity constraints on objects are given in temporal logic. Transactions are specified by pre/postconditions to produce "executable" state sequences. In order to guarantee that executable state sequences already become admissible, integrity constraints are completely transformed into additional pre/postconditions. We introduce general rules for these transformations. Thus, schema specifications can be refined and simplified systematically.

KEYWORDS: database design, conceptual schema, database integrity, dynamic constraints, temporal logic, pre/postconditions

1. INTRODUCTION

Conceptual database design aims at defining a global information structure independently of any implementation [TF82, Ce83]; its result is the so-called conceptual schema. This phase corresponds to the specification phase in general software design meant to bridge the gap between requirements analysis and implementation. If conceptual design is supported by formal specification techniques, it may form a basis of rapid prototyping, too [Fu84].

Usually, the conceptual schema characterizes the structure of the database contents at any single instant, i.e., it defines the common structure of all database states. If a database, however, is considered together with its usage, the behaviour over the course of time also has to be described.

Dynamic database behaviour results from a sequence of database states. Whereas "static" integrity constraints define "consistency" of a single state, "dynamic" integrity constraints are concerned with "admissibility" of a state sequence. To specify admissible state sequences formally, temporal logic is an appropriate language. States within a sequence can conveniently be related by means of temporal formulas which provide special quantifiers like "always ... until" or "sometime ... before" [CCF82, Ku84, ELG84/LEG85].

During runtime of a database, transitions between states are caused by user transactions. To exclude undesirable database manipulations which lead to inconsistent states, a set of basic transactions should be agreed upon during design. Then, all applications have to be composed of them. In the implementation, transactions are realized by programs that access the database system. During design, they can be specified independently of their later implementation by means of pre/postconditions. Thus, the set of possible transaction sequences is restricted to "exe-

cutable" sequences in which each state between two transactions must satisfy the respective post- and precondition.

This specification of transactions completes the original specification of dynamic database behaviour: Only those state sequences are allowed which are induced by executable transaction sequences and which are admissible with respect to the integrity constraints.

In order to ensure correct database behaviour, it becomes necessary to monitor the constraints during runtime. In contrast to expensive universal monitoring mechanisms, transactions offer a possibility of controlling database usage "locally" and therefore efficiently: Checks may be limited to the data actually affected and may be suited to the respective operation. The same principle is persued in [Ni82] to simplify static constraints for standard relational updates.

To incorporate this kind of monitoring, the specification of transactions should be refined in subsequent design steps: Integrity constraints have to be converted into more restrictive pre/postconditions so that executable transaction sequences already guarantee admissible state sequences. A similar procedure is recommended by [VF85], but only for static and transitional constraints (i.e. conditions on single state transitions).

This paper presents a methodology for such a stepwise specification of dynamic database behaviour. Unlike the related literature, we concentrate on dynamic integrity constraints which refer to arbitrary intervals of state sequences and we give rules how to transform constraints into pre/postconditions systematically. In particular, verification of admissibility becomes unnecessary when applying general transformation rules.

The following section introduces conceptual schema specifications and especially explains integrity constraints and transaction specifications. Section 3 describes our method of refining schema specifications by means of transformations. Finally, related work and some conclusions will be discussed.

2. CONCEPTUAL SCHEMA SPECIFICATION

The original specification of a conceptual database schema consists of three levels, each built upon the previous one:

(1) data types
(2) object types
    with integrity constraints
(3) transactions

In the following, we only give a short introduction to schema specifications, but emphasize those aspects which are relevant to the main subject of this paper. More detailed work on levels (1) and (2) can be found in [LEG85].

2.1 DATA AND OBJECTS

Data types are a fixed reservoire of values for the database, e.g., bool, int, no, text, year; they may be defined by, for instance, algebraic equational specifications. Objects are things about which information is to be stored in the database. They are partitioned into sorts, and information about them is carried by functions between objects and data. Predicates on objects are represented as bool-valued functions. For example, in a schema for an "Automobile Registration Authority" similar to that of [ISO82], the following object sorts and functions appear:

```
sorts      CAR, MODEL, CAR-OWNER, ...
subsorts   CAR-OWNER =
             MANUFACTURER+GARAGE+PERSON
             ...
functions  regno:       CAR    -> no
           model-of:    CAR    -> MODEL
           produced:    CAR    -> bool
           registered:  CAR    -> bool
           approved:    MODEL  -> bool
           name:        CAR-OWNER -> text
           owner:       CAR -> CAR-OWNER
           this-year:   -> year
```

Later examples will refer to this syntactic structure.

We assume the reader to be familiar with the construction of predicate calculus formulas from object functions. General temporal formulas can be built from those formulas using

- logical connectives ($\wedge, \vee, \neg, =>, \ldots$)
- quantification over possible objects ($\underline{\forall}, \underline{\exists}$)
- temporal quantification by always, sometime, next
- bounded temporal quantification by always/sometime ... before/until

In order to express dynamic integrity constraints, only the following special kinds of temporal formulas are used:

$$\underline{\forall} x_1{:}s_1 \ \ldots \ \underline{\forall} x_n{:}s_n$$

$$\underline{\text{always}} \ (\ \alpha \ => \ \begin{matrix}\underline{\text{always}} \\ \underline{\text{sometime}}\end{matrix} \ \beta \ \begin{matrix}\underline{\text{before}} \\ \underline{\text{until}}\end{matrix} \ \tau \ )$$

Here α, β, and τ are nontemporal formulas, i.e. formulas without temporal quantifiers, and $x_1,\ldots,x_n$ are all free variables of sorts $s_1,..,s_n$ that occur in these formulas. In the following, a simplified syntax is used for constraints; it omits the standard prefix and allows to omit the α and τ parts:

$$\left[\underline{\text{from}} \ \alpha\right] \left\{\begin{matrix}\underline{\text{always}} & \beta \\ \underline{\text{sometime}} & \beta\end{matrix}\right\} \left[\begin{matrix}\underline{\text{before}} & \tau \\ \underline{\text{until}} & \tau\end{matrix}\right]$$

Examples: The integrity constraint

(I1) from produced(c) $\wedge$ this-year=y
sometime registered(c)
before this-year≥y+2

(with variables c:CAR and y:year) says that each car must be registered in the year of its production or in the following year. The mentioned database object this-year is assumed to contain the current year.

Two further constraints shall restrict the succession of ownership relations. For this purpose, we consider the subsort to which an object of sort CAR-OWNER belongs, i.e. either to MANUFACTURER, GARAGE, or PERSON. Membership in a subsort is expressed by the standard predicate 'is'.

(I2) from owner(c) is MANUFACTURER
sometime owner(c) is GARAGE
before owner(c) is PERSON

(I3) from owner(c)=co $\wedge$ co is GARAGE
always owner(c)=co
before owner(c) is PERSON
(variables: c:CAR, co:CAR-OWNER)

A car must be passed from a manufacturer to a garage first, before it may be passed to a person. It must remain property of the same garage before it is sold to a person. ***

Aside from the constraints above one can imagine more complicated temporal formulas or even other kinds of dynamic assertions which might refer to the past [Ku84]; such constraints are not treated in this work. The restricted formulas from above are not only an appropriate basis of monitoring or refining specifications, but also turn out to be sufficient for many applications.

The object level of the schema specification determines how database states are structured and which state sequences are admissible. In a state σ, the sort and function names are interpreted by finite sets of "actual" objects and by functions between object and data sets. In a state sequence $\underline{\sigma} = \langle\sigma_0,\sigma_1,\sigma_2,\ldots\rangle$, this interpretation may change from state to state. $\underline{\sigma}$ is called admissible iff it satisfies the integrity constraints.

We briefly define what it means that an infinite state sequence $\underline{\sigma}$ satisfies a constraint Γ. Let '$[\sigma,\zeta]\vDash\varphi$' denote the fact that a nontemporal formula φ holds in a single state σ for a substitution ζ. ζ indicates the objects and data which are substituted for the free variables in φ. For substitutions, all "possible" objects must be considered, i.e. all objects that belong to some database state as actual objects.

(1) $\underline{\sigma}$ satisfies
$\Gamma \equiv$ from α always β before τ iff.
For all substitutions ζ of free variables in α,β,τ and for all states

σ_i, $i \geq 0$, the following holds:
If $[\sigma_i, \zeta] \models \alpha$, then $[\sigma_j, \zeta] \models \beta$
for each state σ_j, $i \leq j < \mu$, where
$\mu = \min(\{k \mid k \geq i \wedge [\sigma_k, \zeta] \models \tau\} \cup \{\infty\})$.

Starting from any state σ_i in which the "start condition" α holds, all states following that state (including σ_i itself) must satisfy the condition β, before the "termination condition" τ becomes valid for the first time (in σ_μ). This situation is required for each substitution ζ. If τ does not occur at all, β must hold in the entire future of σ_i ($\mu = \infty$).

(ii) $\underline{\sigma}$ satisfies
$\lceil \equiv \underline{\text{from}}\ \alpha\ \underline{\text{sometime}}\ \beta\ \underline{\text{before}}\ \tau$ iff:
For all substitutions ζ and for all states σ_i, $i \geq 0$, holds:
If $[\sigma_i, \zeta] \models \alpha$, then there is some state σ_j, $i \leq j < \mu$, such that $[\sigma_j, \zeta] \models \beta$ (μ as above).

In contrast to (i), there must exist at least one state following σ_i which satisfies β before the termination condition τ occurs.

(iii) Analogous definitions apply to the combinations of $\underline{\text{always}}$/ $\underline{\text{sometime}}$ with $\underline{\text{until}}$. Here the bound μ must be included.

According to the kind of implicit quantification over states, we distinguish between $\underline{\text{universal constraints}}$ in which the keyword $\underline{\text{always}}$ appears and $\underline{\text{existential}}$ ones with $\underline{\text{sometime}}$.

The above semantics of constraints refers to infinite state sequences which represent the complete behaviour of a database from an initial state (σ_0) to the future. From a practical point of view, it is more interesting to consider properties of the partial database behaviour up to an actual state (σ_n). Therefore, we introduce the following notions.

A finite state sequence $\underline{\sigma} = \langle \sigma_0, \ldots, \sigma_n \rangle$ is called (strictly) $\underline{\text{admissible}}$ iff the infinite sequence $\langle \sigma_0, \ldots, \sigma_n, \sigma_n, \sigma_n, \ldots \rangle$ (remaining constant from position n on) is admissible. If some infinite continuation $\underline{\sigma}$ (not necessarily $\langle \sigma_n, \sigma_n, \ldots \rangle$) exists such that $\underline{\sigma}.\underline{\sigma}$ becomes admissible, then $\underline{\sigma}$ is $\underline{\text{possibly admissible}}$. In this case, $\underline{\sigma}$ is not strictly admissible if and only if the sometime-condition of some existential constraint has not been satisfied up to σ_n, although the start condition had occurred. For example, the condition 'registered(c) in constraint (I1) might still be pending, after 'produced(c)' has become valid.

In order to avoid any checking that looks ahead from the actual state, we proceed on the "optimistic" assumption that all pending conditions are satisfiable in the future. Nevertheless, the user might choose a wrong continuation after the actual state. Formally, we assume that a distinguished (infinite) state sequence $\underline{\lambda}$ exists in which every formula holds. A finite state sequence $\underline{\sigma}$ is defined to be $\underline{\text{provisionally admissible}}$ iff $\underline{\sigma}.\underline{\lambda}$ is admissible. To assure provisional admissibility, it suffices to prove that no constraint has been violated in σ_n or in any state σ_j before σ_n. For instance, an existential constraint is violated in this sense, if its termination condition occurs although the sometime-condition has not been valid before.

The latter kind of admissibility can be controlled by means of an integrity monitor which only refers to the database history, but does not involve any lookahead [LEGB5]. For this reason, that property will be the goal of refining specifications in section 3.

2.2 TRANSACTIONS

The third level of a schema specification describes all transactions that may change the database contents. In the style of Hoare s assertions, transactions are specified by means of pre- and postconditions.

Such conditions may be expressed by arbitrary formulas of predicate logic (nontemporal formulas). A precondition contains a

condition on which the transaction may be executed. Any state directly following the transaction must satisfy the corresponding postcondition. Besides, it is implicitly assumed as a "frame rule" that all object functions that are not affected by the postcondition remain unchanged. To specify different cases of state transitions, several pairs of pre/postconditions may be given for one transaction.

Examples: The automobile registration schema of [ISO82] includes transactions to register or sell a car. They are specified here as follows:

(T1) REGISTER (c:CAR, m:MODEL,...)
pre approved(m)
post registered(c) [=true]
∧ model-of(c)=m ∧ ...

A car c may only be registered, if the declared model m has already been approved. During registration, any information passed as parameter is appropriately stored.

(T2) SELL (c:CAR, newco:CAR-OWNER)
pre --- [true; no restriction]
post owner(c) = newco

A car c is sold to a new owner newco . ***

The frame rule simplifies specifications, because only changes must be mentioned in a postcondition. Since arbitrary formulas are allowed, conditional or alternative effects may be formulated. Thus, the poststate of a transaction may be left indefinite within a certain range, and it will be determined uniquely only by later refinement or implementation.

[Ku84/Ku85a] and [VF85] also use pre/postconditions to specify transactions, but apart from the frame rule they have other implicit assumptions which seem to be more restrictive than helpful in many applications. Besides, [VF85] only admit conjunctions of ground formulas (like in the examples above), thus excluding implications, existence statements, etc. In [VCF81] and [CVF84] different formalisms of specifying transactions and operations are compared by means of examples.

Formally, syntax and semantics of transactions are given by the following definitions:

A transaction specification consists of a name T, (sorted) parameter variables X and several pre/postconditions P_j/Q_j ($j \geq 1$); notation: $\{P_j\}\ T(X)\ \{Q_j\}$. These conditions are nontemporal formulas in which additional free variables Y may occur beside X. A substitution ζ of parameter variables is called an actual parameter.

T transfers a state σ into a state σ' with an actual parameter ζ iff:

(a) There is a precondition P_j and a substitution ξ of the free variables Y such that. $[\sigma, \zeta+\xi] \models P_j$.

(b) For all pre/postconditions P_j/Q_j and for all substitutions ξ of the free variables Y, the following holds:
$[\sigma, \zeta+\xi] \models P_j$ implies $[\sigma', \zeta+\xi] \models Q_j$

(c) σ' is minimally changed to σ', i.e., no change of any object function value from σ to σ' can be undone without violating condition (b)

Part (c) of the definition gives a semantic characterization of the frame rule. The so-called frame of T is constituted by all formulas φ which are invariant in the following sense:

For all triples (σ, σ', ζ) such that T transfers σ into σ' with ζ and for all P_j, ξ as above, the following holds:
$[\sigma, \zeta+\xi] \models P_j \wedge \varphi$ implies $[\sigma', \zeta+\xi] \models \varphi$
(We assume that $\lambda+Y$ already contains the free variables occurring in φ.)

Obviously formulas that do not mention any function used in the postconditions belong to the frame But there may also be conditions in the frame which involve those functions, trivial examples are the postconditions themselves. (Of course, their negations are not invariant.) To analyze specifications, derivation rules for frame formulas are needed.

Transaction specifications restrict possible sequences of transactions or states to "executable" ones.

A transaction sequence $T_1,\ldots,T_n$ is called executable iff there is a sequence $\underline{\sigma} = \sigma_0,\ldots,\sigma_n$ of states, such that σ_{i-1} is transferred into σ_i by T_i with some actual parameter ζ_i (for i=1, ..,n). In this case, the state sequence $\underline{\sigma}$ is also called executable.

Thus, transactions and dynamic integrity constraints can be seen as complementary specifications, together they restrict state sequences to such sequences that are e ecutable and admissible. In the example above, state transitions changing a "car-owner" must be induced by corresponding transactions like SELL (T2), and they must satisfy the constraints (I2) and (I3).

3. REFINING THE SCHEMA SPECIFICATION

The possibility of restricting state sequences by specifying transactions shall be utilized to monitor integrity constraints. The constraints are converted into a refined specification of the transactions so that executable state sequences become provisionally admissible up to each actual state. Thus, dynamic database behaviour is determined by transactions only, and temporal constraints need not be considered for implementation any longer.

This refinement offers the opportunity to observe the impacts of "global" integrity constraints on the "local" usage of the database If new pre/postconditions lead to undesirable restrictions or even to contradictions, the original schema specification should be corrected. Moreover, there is some prospect that integrity monitoring by transactions can be implemented quite efficiently, if checks are suited to the respective kind of change and to the objects affected.

As a preparation for refining a schema specification, the set of object functions is extended by some predicates which keep track of partially satisfied constraints: There may be sometime-conditions which are yet to come or always-conditions which must hold only temporarily. It makes sense to store such "memos" in the database, since they can indicate the actual degree of admissibility to the user.

Then, the pre/postconditions of transactions are completed by further conditions.Semantically, executability of state sequences becomes stronger restricted. At first we show which completion would be necessary without regarding any specific knowledge about a transaction. This transformation corresponds to the universal integrity monitor presented in [LEG85]. By considering which transactions can change which object functions, those completions can be essentially simplified afterwards.

The key idea of monitoring or transforming a dynamic constraint is to split a temporal formula into a nontemporal formula that must hold in the actual state, and a temporal one that remains to be checked in the future. For this purpose, so-called "recursion rules" of temporal logic [MP81] are helpful:

$$\underline{\text{always}}\ \varphi \quad :\Leftrightarrow \quad \varphi \wedge \underline{\text{next}}\ (\underline{\text{always}}\ \varphi)$$
$$\underline{\text{sometime}}\ \varphi \quad :\Leftrightarrow \quad \varphi \vee \underline{\text{next}}\ (\underline{\text{sometime}}\ \varphi)$$

For $\Phi \equiv (\alpha \Rightarrow \underline{\text{sometime}}\ \beta\ \underline{\text{before}}\ \tau)$ and $\Phi \equiv (\underline{\text{sometime}}\ \beta\ \underline{\text{before}}\ \tau)$, the following laws can be derived:

	(1)	(2)
(i) Φ	$\Leftrightarrow (\alpha \vee \neg\tau)$	$\wedge\ (\alpha \wedge \neg\beta \Rightarrow \underline{\text{next}}\ \Phi)$
(ii) Φ	$\Leftrightarrow \neg\tau$	$\wedge\ (\neg\beta \Rightarrow \underline{\text{next}}\ \Phi)$

If one of the temporal formulas Φ or Φ is required for a state sequence, part (1) excludes an illegal occurrence of the termination condition τ in the first state, and (2) indicates on which premises Φ must be satisfied by the rest sequence starting with the next state. According to the interpretation of a constraint "$\underline{\text{from}}\ \alpha\ \underline{\text{always}}\ \beta\ \underline{\text{before}}\ \tau$" (see section 2.1), Φ applies to arbitrary (starting) states and substitutions According to the laws above Φ may apply to following states for certain substitutions; it then replaces the requirement Φ, since Φ implies Φ. This temporary requirement Φ must be recorded in a special memo.

In this section we discuss those two basic

forms of integrity constraints which were explained in section 2.1 explicitly. Each existential constraint

$\Gamma \equiv$ **from** α **sometime** β **before** τ

is transformed in the following way:

Step I:

An additional predicate named sometime β before τ is introduced whose arguments correspond to the free variables in β and τ; in the following, it is abbreviated to SB (though depending on Γ). This predicate must be entered in the database as true for a substitution ζ, if the corresponding condition (Φ from above) remains to be checked with that substitution in the next states; so $SB(\zeta)$ plays the role of the mentioned "memo". Recall that a substitution just represents a certain combination of data and objects to be used as actual arguments.

Thus, as many predicates are added to the schema as constraints are given in the original specification. In special cases, such predicates can be expressed in terms of existing functions and predicates so that a new predicate can be avoided. This will be illustrated in a later example.

Step II:

Let an arbitrary transaction T be given with an original specification or with a specification resulting from transformation of other constraints. Each assertion $\{P\}\ T\ \{Q\}$ consisting of a pair of pre/postconditions P/Q is rewritten to:

$$\{P \wedge \neg SB\}\ T\ \{Q \wedge \underbrace{(\alpha \Rightarrow \neg\tau)}_{(1)} \wedge \underbrace{(\alpha \wedge \neg\beta \Leftarrow SB)}_{(2)}\}$$
$$\{P \wedge SB\}\ T\ \{Q \wedge \neg\tau \wedge (\beta \Rightarrow \neg SB)\}$$

Here, the laws (1) and (11) from above are utilized to restrict the set of post-states. The new specification expresses the requirement that T has to monitor the given constraint as far as this can be done by inspecting the new actual database state. A distinction is made, whether or not a memo SB "exists" previously. The $\Rightarrow$-direction of condition (2) controls "insertion" (SB) or "deletion" ($\neg$SB) of the memo. The $\Leftarrow$-direction of (2) determines if $\neg$SB or SB is retained. It should be noticed that the free variables in the added parts must be named differently from parameter and other variables, because those conditions shall hold for arbitrary objects.

Since postconditions affect only states following transactions, the addition in the first line above has to hold in the initial state of the database, too:

$$(\alpha = \neg\tau) \wedge (\alpha \wedge \neg\beta \Rightarrow SB)$$

Example. By transforming the integrity constraint

(I1) **from** produced(c) $\wedge$ this-year=y
sometime registered(c)
before this-year$\geq$y+2

the following new specification is obtained for the transaction (T1) REGISTER(c,m,...) of section 2.2:

pre_1 approved(m) $\wedge$ $\neg$SB(c ,y)
$post_1$ registered(c) $\wedge$ ...
$\wedge$ ((produced(c) $\wedge$ this-year=y)
$\Rightarrow$ $\neg$this-year$\geq$y +2) } (-)
$\wedge$ ((produced(c) $\wedge$ this-year=y
$\wedge$ $\neg$registered(c))
$\Rightarrow$ SB(c ,y))

Obviously (-) is a tautology that may be dropped

pre_2 approved(m) $\wedge$ SB(c ,y)
$post_2$ registered(c) $\wedge$ ...
$\wedge$ this-year$<$y +2
$\wedge$ (registered(c) $\Leftarrow$ $\neg$SB(c ,y))

For design purposes, it is recommended to introduce a more application oriented name for the predicate SB. A conversion of its parameters may be included, as it can be done for (I1):

to-be-registered-before: CAR **year** $\rightarrow$ **bool**
where to-be-registered-before(c ,y +2)
= SB(c ,y) ***

Step III:

Now, the transaction specifications of step II can be essentially simplified.

Due to the frame rule with respect to the original specification many transactions will not change the added conditions at all, others will not do so for many substitutions. Therefore, the transformation above needs only be done for transactions which possibly change the truth value of α, β or τ, i.e., some of these formulas or their negations are not in the frame of the transaction. Moreover, the new variables of step II may be replaced by those parameter variables, original variables, constants or terms which are involved in changes of the corresponding formula parts. If not all three constraint components change, whole factors of the above conjunctions may be omitted. Finally, the frame rule may be utilized for the new specification, too

To discuss typical situations, let us consider transactions T which change at most one of the constraint components α, β or τ from false to true. Equivalently, either $\neg\alpha$, $\neg\beta$ or $\neg\tau$ is not in the frame of T. The following specifications result from step *III*:

(i) If T can change only α to true:
{P∧¬SB} T {Q ∧ ¬τ ∧ (¬β <= SB)}
{F∧ SB} T {Q}

(ii) If T can change only β to true:
{F∧¬SB} T {Q ∧ ¬SB}
{F∧ SB} T {Q ∧ ¬SB}
or combined: {P} T {Q ∧ ¬SB}

(iii) If T can change only τ to true:
{F∧¬SB} T {Q ∧ ¬α ∧ ¬SB}
{P∧ SB} T {Q ∧ false} (-)
Since the second postcondition (-) cannot hold in any state, the whole line may be dropped so that the precondition of T is restricted to {F∧¬SB} only.

Examples: The above refinement of the specification for REGISTER can be simplified according to case (ii), because only ¬β ≡ ¬registered(c) is not invariant:

{approved(m)} REGISTER(c,m,..)
{registered(c) .. ∧ ¬SB(c,y)}

After execution of REGISTER(c,m,...) no memo may exist for the car c. An implementation of the transaction has to take care of deleting a memo which possibly existed before.

Let a further transaction NEXT-YEAR be given which updates the current year this-year :

{this-year=x} NEXT-YEAR {this-year=x+1}

Referring to constraint (I1), only the termination condition $\tau \equiv$ this-year≥y+2 is changed for y= x-1, so that transformation according to case (iii) and obvious simplification of the resulting postcondition yield:

{this-year=x ∧ ¬SB(c,x-1)}
NEXT-YEAR {this-year=x+1}

Thus, right before execution of NEXT-YEAR any memo from the previous year must not exist any longer. In a way, "the clock must be stopped" in order to settle the required registrations for all cars c with memo SB(c,x-1). ***

All simplifications in step III presuppose that any partial condition or substitution which is excluded here has been checked after its last change (and in the initial state). This is the case if <u>all</u> transaction specifications are treated according to the rules above.

As can be seen by induction on state sequences, the transformation rules do guarantee that executable state sequences satisfy the given constraint in the sense of provisional admissibility. Thus, single verifications are rendered superfluous. Additionally, our examples demonstrate how conditions can typically be simplified further in concrete situations, so that a "readable" specification results in spite of formal transformation.

The transformation of <u>universal constraints</u> will only be given by analogy to existential ones. For each constraint

<u>from</u> α <u>always</u> β <u>before</u> τ

a predicate always β before $\tau \equiv AB$ is needed (step I). Then, the basic transformation of {F} T {Q} reads as follows:

$\{P \wedge \neg AB\}\ T\ \{Q \wedge (\alpha \wedge \neg\tau \Rightarrow \beta) \wedge (\alpha \wedge \neg\tau \Leftarrow AB)\}$

$\{P \wedge AB\}\ T\ \{Q \wedge (\neg\tau \Rightarrow \beta) \wedge (\tau \Leftarrow \neg AB)\}$

This specification is constructed similarly to step II for existential constraints. The subsequent simplifications of step III apply here, too.

Example. The following constraint is to be transformed:

(I3) from owner(c)=co ∧ co is GARAGE
always owner(c)=co
before owner(c) is PERSON

Among the given transactions only SELL (T2) is affected by a transformation, since no other transaction changes the function owner . After some equivalence conversions of the basic transformation a refined specification is obtained for SELL(c,newco) :

pre_1 ¬AB(c,co)
$post_1$ owner(c)=newco
∧ ((owner(c)=co ∧ co is GARAGE) ⇔ AB(c,co)) } (*)

or:

$post_1$ owner(c)=newco
∧ (newco is GARAGE ⇒ AB(c,newco))

($\{pre_1\}\ T\ \{post_1\}$ is equivalent to $\{pre_1\}\ T\ \{post_1\}$, since all omitted assertions are in the frame of T wrt $post_1$.)

pre_2 AB(c,co)
$post_2$ owner(c)=newco
∧(¬newco is PERSON ⇒ co=newco) (+)
∧(newco is PERSON⇔¬AB(c,co))

Thus, restrictions on succession of ownerships have been incorporated into the transaction specification. Subsequent transformation of the other integrity constraint (I2) would lead to a further differentiation. ***

The second postcondition of the example ($post_2$) strongly restricts the possibility of transferring states by this transaction: For some states satisfying the precondition, there is no state satisfying the postcondition. This specifies correctly that certain state sequences are not executable, but the situation seems to be modelled in a somehow complicated way. To get a clearer specification it is recommended that the precondition should exclude cases when no poststate exists.

Example, cont'd.: In spite of the precondition, part (+) of the postcondition cannot be satisfied for arbitrary parameters newco . In this example, it is enough to put this restriction into the precondition in its original form, since no object function being subject to change is involved in this formula. Then, the second specification part of SELL(c,newco) reads:

pre_2 AB(c,co)
∧(¬newco is PERSON ⇒ co=newco) (+)
$post_2$ owner(c)=newco
∧(newco is PERSON⇔¬AB(c,co))

Moreover, here the new predicate AB can be expressed by means of existing predicates provided that the function owner is affected by SELL only. It can be proved by induction on executable state sequences that the following equivalence holds always:

(owner(c)=co ∧ co is GARAGE) ⇒ AB(c,co)

(This was mentioned in the first postcondition $post_1$ as (*).) ***

Static integrity constraints β which correspond to special universal constraints

from true always β before false

do not require introducing any additional predicate, either. For them, transformation of $\{P\}\ T\ \{Q\}$ yields:

$\{P \wedge \beta\}\ T\ \{Q \wedge \beta\}$

In particular, this refinement might give rise to checking the original postcondition Q for consistency with the static constraint β.

Of course, one can imagine situations where certain integrity constraints can be concluded directly from the original transaction specifications so that no transformation is necessary. In any case, general transformations are possible and reliable; thus, they can serve the designer at least as a guideline to refining the schema specification. Besides, the exam-

ples show that systematic simplifications can lead to formulations easy to survey and to grasp, where verification has already been done. So transaction specifications become a sufficient and useful interface to the implementor.

4. RELATED WORK AND CONCLUSIONS

The literature based upon similar calculi (temporal logic and pre/postconditions) contains first approaches to comparable multi-level database specifications [CCF82/ CVF84/ VF85, ku84/ ku85b, SFNC84]. Requirements and methods of verifying temporal constraints are analyzed presuming that complete descriptions of transactions are given beforehand. Already [GM79] have studied verification of static constraints by checking invariant pre/postconditions of transactions. Besides, [ku84/ku85a] is concerned with consistency of schema specifications.

Only [VF85] give some basic methodical hints that the specification - and also the selection - of transactions should be refined in a stepwise way. The above approaches, however, cannot consider "sometime" constraints in refinements, although they may be formulated partially; for this, our notion of "provisional admissibility" proves to be appropriate. Transformation methods for dynamic integrity constraints have not yet been discussed by other authors.

The specification method presented here depends on the principle to fix a module of basic transactions for all database updates. This module becomes responsible for the correctness of dynamic database behaviour. Its design is based on the transformation of the original schema specification: the designer is guided to recognize the impacts of the global specification on the transactions.

The design procedure itself might be supported by an interactive system that controls and manages stepwise transformations and that offers simplifications of logical formulas. Moreover, those transformations may be used to generate a prototype of a database application system from its original specification. As shown by [VF85], preliminary implementations of transactions can be derived from certain restricted pre/postconditions automatically.

Concerning theoretical foundations, the simplification step (III) could be treated more formally than in this paper, but our emphasis was on introducing typical elements of a specification methodology. For simplifications, inference rules are needed to derive frame formulas and to detect irrelevant variable substitutions. The approach of [Ni82] on static constraint simplification should be examined if it can be extended to general transaction specifications. In addition, new monitoring principles for dynamic constraints [LSE86] might be utilized to transform a larger class of temporal formulas.

REFERENCES

Abbreviations:

PODS	Proc. ACM Symp. on Principles of Database Systems
SIGMOD	Proc. Int. ACM-SIGMOD Conf. on Management of Data
IFAIS	Proc. IFIP Work.Conf.on Theoretical and Formal Aspects of Information Systems (A.Sernadas et al., eds.), North-Holland, Amsterdam 1985
VLDB	Proc. Int. Conf. on Very Large Data Bases

[CCF82] Castilho,J.M.V.de/ Casanova,M.A./ Furtado,A.L.: A Temporal Framework for Database Specifications. VLDB 1982, 280-291

[CVF84] Casanova, M.A. / Veloso, P.A.S. / Furtado,A.L.: Formal Database Specification - An Eclectic Perspective. PODS 1984, 110-118

[Ce83] Ceri,S. (ed.) : Methodology and Tools for Data Base Design. North-Holland, Amsterdam 1983

[ELG84] Ehrich, H.-D. / Lipeck, U.W. / Gogolla, M. : Specification, Semantics and Enforcement of Dynamic Database Constraints. VLDB 1984, 301-308

[Fu84] Furtado, A.L. : An Informal Approach to Formal Specifications. SIGMOD Record 14,1 (1984), 45-54

[GM79] Gardarin,G./Melkanoff,M.: Proving Consistency of Database Transactions. VLDB 1979, 291-298

[ISO82] ISO/TC97/SC5/WG3 : Concepts and Terminology for the Conceptual Schema and the Information Base. (J.J. van Griethuysen, ed.) 1982

[Ku84] Kung,C.H. : A Temporal Framework for Database Specification and Verification. VLDB 1984, 91-99

[Ku85a] Kung,C.H. : A Tableaux Approach for Consistency Checking. TFAIS 1985, 191-210

[Ku85b] Kung,C.H. : On Verification of Temporal Database Constraints. SIGMOD 1985, 169-179

[LEG85] Lipeck, U.W. / Ehrich, H.-D. / Gogolla,M. : Specifying Admissibility of Dynamic Database Behaviour Using Temporal Logic. TFAIS 1985, 145-157

[LSE86] Lipeck,U.W./Saake,G./Ehrich,H.D.: Monitoring Dynamic Database Integrity by Transition Graphs. 1986 (submitted for publication)

[MP81] Manna,Z./Pnueli,A.: Verification of Concurrent Programs: The Temporal Framework. in: The Correctness Problem in Computer Science (R.S. Boyer/ J.S. Moore, eds.), Academic Press, 1981, 215-273

[Ni82] Nicolas,J.-M.: Logic for Improving Integrity Checking in Relational Data Bases. Acta Informatica 18 (1982), 227-253

[SFNC84] Schiel,U./ Furtado,A.L./ Neuhold, E.J./Casanova,M.A.: Towards Multi-Level and Modular Conceptual Schema Specifications. Information Systems 9 (1984), 43-57

[TF82] Teorey,T.J./ Fry,J.P.: Design of Database Structures. Prentice-Hall, Englewood Cliffs 1982

[VCF81] Veloso,P.A.S./ Castilho,J.M.V.de/ Furtado,A.L.: Systematic Derivation of Complementary Specfications. VLDB 1981, 409-421

[VF85] Veloso,P.A.S./ Furtado,A.L.: Towards Simpler and Yet Complete Formal Specifications. TFAIS 1985, 175-189

ACKNOWLEDGEMENT

I wish to thank Klaus Drosten for helping me to improve the English of this paper.