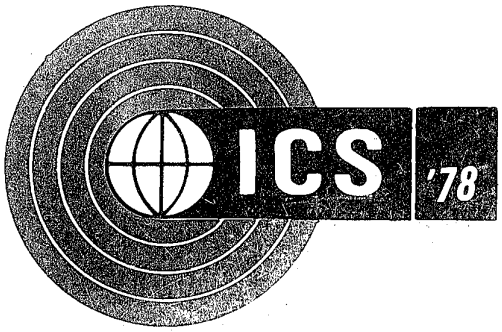

一九七八年國際計算機會議論文集

第一册

PROCEEDINGS OF INTERNATIONAL COMPUTER SYMPOSIUM 1978

VOLUME ONE

PROCEEDINGS

OF

INTERNATIONAL COMPUTER SYMPOSIUM 1978

DECEMBER 18–20, 1978

NANKANG, TAIPEI, REPUBLIC OF CHINA

VOLUME ONE

TO CELEBRATE THE 50TH ANNIVERSARY

OF ACADEMIA SINICA

ACADEMIA SINICA PUBLICATION

INTERNATIONAL COMPUTER SYMPOSIUM 1978

December 18–20, 1978

Nankang, Taipei, Republic of China

HONORARY CHAIRMAN

DR. S. L. CHIEN

PRESIDENT, ACADEMIA SINICA, R.O.C.

GENERAL CHAIRMAN

DR. JULIUS T. TOU

Member of Academia Sinica, R.O.C.

University of Florida, U.S.A.

ADVISORS

L. A. CHEN	Vice Minister of Education, R.O.C.
HOWELL S. C. CHOU	President, Computer Society of The Republic of China
YAOHAN CHU	Professor, University of Maryland, U.S.A.
H.C. FANG	President, Industry Technology Research Institute, R.O.C.
I. T. HO	Senior Engineer, IBM, U.S.A.
N. H. KUO	Dean, College of Engineering, National Chiao Tung University, R.O.C
K. C. LEE,	Director, EDP Center, DGBAS, Executive Yuan, R.O.C.
S. S. SHU	Chairman, National Science Council, R.O.C.
C. C. YU	Dean, College of Engineering, National Taiwan University, R.O.C.

Proceedings of International Computer Symposium 1978 (Vol. 1)

SPECIFYING EXTERNAL DATA BASE SCHEMAS

A. L. Furtado
Departamento de Informatica
Pontificia Universidade Catolica do Rio de Janeiro

ABSTRACT

A view construct is proposed which consists of a derived relation and the opera - tions defined on it. Attention is focussed on the update operations, for which the conditions, effects, and side effects must be specified.

In order to guarantee that constraints are preserved, it is proposed that all sets of views, constituting the external schema of each user or class of users, be specified (and revised when needed) together, by the enterprise administrator in consultation with the application administrators.

A simple data base environment is described and used to illustrate the suggested approach.

1. INTRODUCTION

In this research we assume that a data base exists as a set of normalized base relations [1]. On every such relation the following primitive operations are available:

- interrogate, which yields the values of attributes of indicated tuples of the relation;
- insert, which adds tuples to the relation;
- delete, which removes indicated tuples from the relation;
- modify, which changes the values of attributes of indicated tuples of the relation.

The last three are the update operations. They must be used with caution bacause their effect is to introduce changes to the data base. They have no effect however if certain simple restrictions are violated: insert fails if duplicate tuples would appear in the relation; delete and modify fail if the indicated tuples are not found in the relation.

Usually other more complex restrictions are recognized for specific data bases: integrity and authorization constraints [2,3]. It is in general difficult and costly to ensure that users do not violate these constraints when updating the data base.

Our approach is to present to each user or class of users, as his external schema [4], a set of views. Our notion of a view refers not only to a relation derived from base relations (cf. for example [1,5]) but also to the operations defined on this derived relation.

This research has been sponsored by the Conselho Nacional de Desenvolvimento Cientifico e Tecnologico. Helpful comments from K. Sevcik and C. Saraiva dos Santos are gratefully acknowledged.

One of these operations will be, for all views, the interrogate operation, which is thus considered to be inherited from the base relations level, as we shall now indicate. Let X_V be an expression determining how the derived relation in a view V is to be obtained from given base relations. An interrogate on V would then be translated into the (primitive) interrogate operation on the underlying base relations, using X_V to guide this translation.

We shall not use the same strategy for the update operations. Each update operation defined in a view V will have a name, and its expression will include one or more explicit calls to the primitive update operations to be executed on base relations (some of these base relations may be other than the underlying ones, as we shall see next).

The purpose of these more elaborate "higher-level" update operations is to enforce the constraints in different ways that take advantage of the characteristics of each operation. The expression of an operation includes, besides the effects directly intended by the user, the conditions that must hold for the operation to be executed and the side effects of the operations. Side effects are updates not directly intended by the user. An operation σ must be designed in such a way that, if the conditions P_σ hold then the execution of the effects E_σ and side effects S_σ can be proved to preserve the set of all constraints C, or in the notation of [6]:

$$P_\sigma \{E_\sigma , S_\sigma\} C$$

Both conditions and side effects may refer to base relations that do not contribute to the derivation of V.

The point is that if all update operations are so designed we shall never have to check the constraints directly. Also further advantage can be taken of the fact that the update operations are not independent in general; often the execution of an operation σ affects positively the conditions $P_{\sigma'}$ for an operation σ' to be executed. In such cases, if it is costly to test our initially specified $P_{\sigma'}$, we can redesign it to involve the verification of the favorable effects or side effects of σ (whose presence imply that the initially designed $P_{\sigma'}$ is true).

It should be clear that the proposed approach relies on the assumption that all the sets of views in a data base are designed together, by the same person or group of persons, and that the sets of views remain relatively stable. They should be designed together because of their interdependence and the possibility of unwanted interference [7] among users. The persons designing the views in an integrated way would be the enterprise administrator in consultation with the application administrators [4], not the users themsemves; note that a user may not be interested (or authorized)to know of certain conditions and side effects of operations. The relative stability is to be expected from the ample freedom left to interrogates (for accommodating unanticipated queries) within the derived relations in the views. It appears to be a realistic decision to restrict the updates more severely, and require that each new type of update operation be discussed with respect to the existing ones [8].

As has been shown, our notion of views extends the usual one. For us a view is not only a "window" [5] to give access tc information, but also a "shade" to hide information

that is irrelevant or not authorized to the user, and a "screen" to prevent illegal updates [3].

In this paper this notion of views will be developed, with a particular emphasis on the update operations. Section 2 describes our view construct as a convenient programming languages feature. Section 3 describes a simple data base to be used for illustrating the proposed approach. Section 4 presents the views designed for the example data base. Section 5 considers briefly the case where a user is both interested in and authorized to find out about the conditions and side effects involved in update operations. Section 6 contains the conclusions.

2. THE VIEW CONSTRUCT

As indicated, views are to be designed by the enterprise administrator in consultation with the application administrators. The design involves two parts:

- the specification of the derived relation in terms of its name and attributes, and of the update operations in terms of their names and parameters; this part is visible to the users, and corresponds to the displays section in the view construct below;
- the definition of the derived relation in terms of its underlying base relations, and of the update operations in terms of their conditions, effects and side effects with respect to the underlying base relations and possibly other base relations (involved in the conditions and side effects); this part is in principle not visible to the users, and corresponds to the expressions section in the view construct below.

We now introduce the structure of the view construct :

```
view  name
      displays
          tuples      attributes of the tuples of
                      the view
          operations  names and parameters of
                      each update operation

      expressions
          tuples      how the tuples of the view
                      are derived from the tuples
                      of base relations
          operations  for each operation, its
                          conditions
                          effects
                          side effects
end   name
```

An interrogation operation is assumed to be available in all views, referring in unrestricted ways to the view attributes appearing in the displays section. The primitive interrogate, insert, delete, and modify operations are not directly available to the users being utilized only inside the expressions section.

In this paper we shall not commit ourselves to any formal syntax for the design of views. In the examples of section 3 we shall employ a quasi-natural language; in practice, some language in this general style seems to be appropriate for the design phase, especially when recalling that the administrators in charge will not have a professional programming background as a rule.

After designing all the views it is necessary to verify that the update operations indeed preserve the constraints, are mutually compatible, and are sufficient to handle the data base [8].

Then comes an implementation phase which may proceed through top-down refinement from the views written in quasi-natural language. This phase of course does not require the participation of the administrators above, except that it is necessary that the implementors verify (and pass some evidence to the administrators) that as finally programmed the views correspond to the design. Also, the data base administrator might be consulted with regard to the efficiency of the implementation.

Our view construct is comparable to other constructs such as clusters, forms, and modules [9,10,11,12,13]. On the other hand, recent work in data base languages seems particularly compatible with lines followed here [14]. Our operations are like the functions in [13] in style.

Once implemented, a view can be stored in a library [9] and made available to the authorized user or class of users. Some sort of open statement may bring it from the library into the users applications programs (the "client" modules) [11]. We expect that the different applications program that will appear, be changed, or vanish during a data base's lifetime will have their needs satisfactorily met by a set of views that will remain relatively stable (some problems related to changes will be briefly discussed in the context of the example in section 4).

After opening a view in his program, a user can declare one or more variables bound to the view. Here there is a major difference between the usual typed variables and the typed variables for shared data [15]. The latter instead of values of the type given in the declaration contain references to the shared objects containing the values. Such references are elsewhere called cursors [1]; to the treatment of cursors in [1] all we have to add here is that besides the explicit cursor available to the user, allowing him to refer to tuples in the view, there must occur the automatic creation and setting of one "internal" cursor per base relation in the view.

This recalls that views are "virtual" objects that are materialized through the base relations, and that updates on one view affect other views derived from common base relations. The fundamental consequence of this fact is that the behavior of a view cannot be explained solely by the operations defined on it; tuples appear, are changed, or vanish from the view due to operations defined in other views. Thus a view cannot be considered in isolation as an intelligible unit of study, which sharply distinguishes views from other constructs for incorporating "abstractions". This argument leads again to the necessity of designing together the collection of all the sets of views given to each user, which constitute the entire data base.

3. A SIMPLE DATA BASE ENVIRONMENT

As an example data base environment, we consider the personnel segment of a small manufacturing enterprise. While the example is intended to suggest realism, it is highly simplified, and certainly does not cover the breadth of situations that may arise in more detailed enterprise descriptions.

A. The Base Relations

The attributes treated in our example are:

N - name of employee
S - salary

J - job title
K - skill
T - task
P - project
L - leader of a project

The base relations which represent the relationships among entities with these attributes are :

EMP(N,S,J) - employee's name, salary and job title
REQ(T,K) - requirement of a skill to do a task
ASN(N,T,P) - assignment of an employee in a project to a task
MNG(P,L) - management of a project by a leader
CAP(N,K) - capabilities (skills) possessed or acquired by employees

Figure 1 indicates the presence of attributes in relations diagramatically.

about these special employees.

C. The Activities of the Enterprise

The personnel manager hires employees by associating a salary (at least the minimum wage), and a job title with their name, and may fire employees, but only if they are not currently assigned to any project.

The engineering manager initiates new projects by specifying their names and the name of the initial leader of each. He may replace the leader of a project, or suspend a project by leaving it with no leader. No employees may continue to be assigned to a project that is suspended.

A suspended project may be permanently terminated by the engineering manager, or may be restarted by assigning a new leader.

Various tasks compose each project, and the engineering manager is responsible for indicating what skills are required of an em-

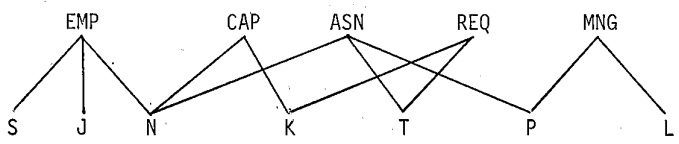

Figure 1. The base relations and their attributes

B. The Users

The users authorized to perform update operations are the employees holding the positions

personnel manager,
engineering manager,
training manager,
and leader of some project.

Relation EMP does not hold information

ployee to perform each task. The engineering manager also associates employees with projects (but not suspended ones), and terminates such associations.

Employees acquire new skills through training, but lose old skills through lack of use or changing technology. The training manager is responsible for recording the skills currently possessed by each employee.

Each leader of a project determines how employees associated with his project are assigned to tasks. An employee must possess all the skills required for each task assigned to him.

D. Constraints

From the description of the activities of the enterprise, we can reasonably formulate a number of constraints:

1. Salaries must be at least equal to the minimum wage
2. A hired employee must have exactly one salary and job title at a time
3. Only hired employees can be associated with projects
4. Only hired employees can have their skills recorded in the data base
5. A project can have at most one leader at a time
6. A project must have an initial leader when it is created
7. Only projects without a leader can be terminated
8. Employees can only be associated with projects that currently have a leader
9. Only employees that are not currently associated with any project can be fired
10. To perform a task an employee must have all the skills required for the task.

There are different ways to express the same constraint. For example, constraint 5 could be phrased in terms of the functional dependence P → L, noting also that the relationship between these two attributes is partial (i.e. the leader stays "undefined" while a project remains suspended). Constraint 3 can be expressed as : N in ASN is a subset of N in EMP. Constraint 10 is also expres - sible in terms of subsets, in a slightly more complex way involving partitioning (grouping tuples by equality with respect to certain attributes).

4. DESIGNING THE EXTERNAL SCHEMAS

We shall now, for the simple example of the previous section, draw the external schemas of the recognized users. First we intro - duce the few conventions adopted, and then, for each user, we shall exhibit his schema by way of a diagram, give the meaning of the views in the schema, and then develop the views according to the framework introduced in Section 2; we give all views because, as argued, their design must be interdependent, but the reader may choose to examine only a few as examples.

Finally, some remarks are made to illustrate how the constraints are incorporated into the update operations, and how the design reacts to certain changes.

A. Conventions

Each view was given a distinct name, regardless of the possibility that two views in the schemas of different users might be identical. Note however that the characterization of views includes the allowable operations on them; in this sense no two views in our example are identical. In particular, in the diagrams, dotted lines lead to attributes that can only be interrogated.

For convenience we gave to the attributes the same names used for the related attributes in the base relations. One might prefer to assign names more meaningful to each user.

As said, the primitive operations for base relations appear in the operations ex - pressions; however the interrogate operation, needed for retrieving attribute-values of tuples existing in the base relations, will

not be mentioned explicitly. As to the primitive update operations they will be mentioned explicitly; the reader is asked to note that in some cases they will affect more than one base relation tuple.

We shall not indicate what happens if conditions for an operation are not fulfilled, some general "fail" notice being assumed.

The notation "*" will be used for the undefined value, and "-" is a place-holder standing for any value of the corresponding attribute.

The "macro" delete' (n,t,p), appearing in the expressions of a number of operations, stands for :

```
if (n,t,p) is the only tuple in ASN
   with the given n and p
then modify (n,t,p) into (n,*,p) in ASN
else delete (n,t,p) from ASN
```

This strategy prevents the detaching of an employee from a task in a project from destroying the information that he is still associated with the project (even when not attached to any task in the project).

B. Personnel Manager's Schema

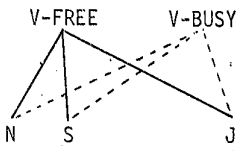

Figure 2. The personnel manager's schema

V-FREE(N,S,J) - employees not associated with projects

V-BUSY(N,S,J) - employees associated with at least one project

```
view V-FREE
   displays
      tuples (N,S,J)
      operations hire (n,s,j)
                 fire(n)
   expressions
      tuples from EMP such that n does not
             appear in ASN
      operations - hire
                   conditions: no tuple in
                     EMP with n, and
                     s ≥ min.wage
                   effects:insert (n,s,j)
                     into EMP
                   side effects: none
                 - fire
                   conditions: none
                   effects: delete (n,-,-)
                     from EMP
                   side effects: delete (n,-)
                     from CAP
end V-FREE

view V-BUSY
   displays
      tuples (N,S,J)
   expressions
      tuples from EMP such that n appears
             in ASN
end V-BUSY
```

C. Engineering Manager's Schema

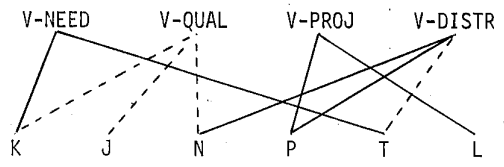

Figure 3. The engineering manager's schema

```
V-NEED(T,K)    - requirement of a skill to do
                 a task
V-QUAL(N,J,K)  - qualifications of employees
V-PROJ(P,L)    - projects and their leaders
V-DISTR(N,T,P) - distribution of employees to
                 projects and tasks

view  V-NEED
      displays
         tuples (T,K)
         operations require (t,k)
                    remove (t,k)
      expressions
         tuples from REQ
         operations - require
                      conditions none
                      effects insert (t,k)
                         into REQ
                      side effects if (n,t,p)
                         for some n and p is
                         in ASN and (n,k) is
                         not in CAP then
                         delete'(n,t,p) from
                         ASN
                    - remove
                      conditions none
                      effects delete (t,k)
                         from REQ
                      side effects none
end   V-NEED

view  V-QUAL
      displays
         tuples (N,J,K)
      expressions
         tuples from EMP and CAP concatenat-
                ing tuples with same n
end   V-QUAL

view  V-PROJ
      displays
         tuples (P,L)
         operations initiate (p,ℓ)
                    replace (p,ℓ)
                    suspend (p)
                    restart (p,ℓ)
                    terminate (p)
      expressions
         tuples from MNG
         operations - initiate
                      conditions: there is no
                        tuple in MNG with p
                      effects: insert (p,ℓ)
                        into MNG
                      side effects: none
                    - replace
                      conditions: none
                      effects: modify (p,-)
                        into (p,ℓ) in MNG
                      side effects: none
                    - suspend
                      conditions: none
                      effects: modify (p,ℓ)
                        into (p,*) in MNG
                      side effects: delete
                        (-,-,p) from ASN
                    - restart
                      conditions: there is a
                        tuple (p,*) in MNG
                      effects: modify (p,*)
                        into (p,ℓ) in MNG
                      side effects: none
                    - terminate
                      conditions: (p,*) is in
                        MNG
                      effects: delete (p,*)
                        from MNG
                      side effects: none
end   V-PROJ
```

```
view  V-DISTR
      displays
         tuples (N,T,P)
         operations associate (n,p)
                    disassociate (n,p)
      expressions
         tuples from ASN
         operations - associate
                      conditions: there is a
                        tuple with n in EMP
                        and a tuple (p,t) for
                        some t in MNG and
                        there is no tuple
                        (n,-,p) in ASN
                      effects: insert (n,*,p)
                        into ASN
                      side effects: none
                    - disassociate
                      conditions: none
                      effects: delete (n,-,p)
                        from ASN
                      side effects: none
end   V-DISTR
```

D. Training Manager's Schema

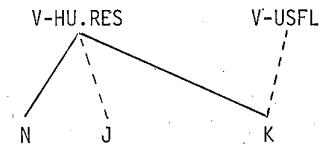

Figure 4. The training manager's schema

V-HU.RES(N,J,K) - human resources

V-USFL(K) - useful skills, i.e. skills required by at least one task.

```
view  V-HU.RES
      displays
         tuples (N,J,K)
         operations acquire (n,k)
                    lose (n,k)
      expressions
         tuples from EMP and CAP concatenat-
           ing tuples with same n
         operations - acquire
                      conditions: there must
                        be a tuple with n in
                        EMP
                      effects: insert (n,k)
                        into CAP
                      side effects: none
                    - lose
                      conditions: none
                      effects: delete (n,k)
                        from CAP
                      side effects: if (t,k)
                        for some t is in REQ
                        and (n,t,-) is in
                        ASN then delete'(n,
                        t,-) from ASN
end   V-HU.RES

view  V-USFL
      displays
         tuples (K)
      expressions
         tuples from REQ taking only attribute K
end   V-USFL
```

E. Project Leaders' Schemas

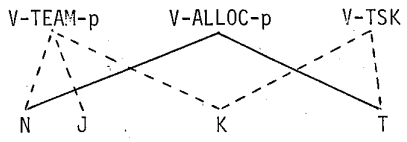

Figure 5. The external schema of the leader of each project p

V-TEAM-p(N,J,K) - employees in project p
V-ALLOC-p(N,T) - allocation of employees to tasks in project p
V-TSK(T,K) - requirement of a skill to do a task

view V-TEAM-p
displays
tuples (N,J,K)
expressions
tuples from EMP and CAP concatenating tuples with same n and taking only the tuples such that (n,-,p) is in ASN and (p,ℓ) is in MNG, ℓ being the user
end V-TEAM-p

view V-ALLOC-p
displays
tuples (N,T)
operations assign (n,t)
release (n,t)
expressions
tuples from ASN taking only the tuples (-,-,p) such that (p,ℓ) is in MNG, ℓ being the user,and taking only the attributes N,T
operations - assign
conditions: for every k in tuples (t,k) in REQ there must be (n,k) in CAP, and there must be some tuple (n,-,p) in ASN
effects: if (n,*,p) is in ASN then modify (n,*, p) into (n,t,p)else insert (n,t,p) into ASN
side effects: none
- release
conditions: none
effects: delete'(n,t,p) from ASN
side effects: none
end V-ALLOC-p

view V-TSK
displays
tuples (T,K)
expressions
tuples from REQ
end V-TSK

F. Remarks

As an example of how constraints are expressed in terms of conditions and side-effects, consider the following constraint, stated in section 3 :

10. To perform a task an employee must have all the skills required for the task.

This constraint involves base relations CAP, REQ, and ASN. The procedure below gives an idea of the work required for checking this constraint directly :

a. group the tuples of CAP by N and in each group consider the respective set of skills; this gives the sets of skills possessed by each employee;

b. group the tuples of REQ by T and in each group consider the respective set of skills; this gives the sets of skills required for each task;

c. form the pairs (n,t) such that the set of skills of employee n contains the set of skills required for task t; this gives all potential assignments of employees to tasks that would be legal;

d. verify if the set of pairs (n,t) taken from ASN is contained in the set of pairs (n,t) obtained in the previous item; this shows if the actual assignments are legal.

Of course it is much better to do a selective checking tailor-made for each operation that might violate the constraint. In our case this is done as follows:

- for the assign operation: condition: check if the set of skills required for the (indicated) task is contained in the set of skills possessed by the employee;
- for the require operation: side-effect: delete all assignments to the task of employees not having the skill being required;
- for the lose operation: side-effect: delete all assignments of the employee to tasks that require the lost skill;

and then we note that although remove and acquire affect the base relations involved they cannot violate the constraint and thus no conditions or side-effects are needed. Also, the associate operation, which in a similar way to assign causes an insertion into ASN, cannot violate the constraint, since its effect is to insert tuples of the form (n,*, p), with an "undefined" task.

Another given constraint :

9. Only employees that are not currently associated with any project can be fired.

is enforced in the operation fire without explicit reference to conditions or side effects. In this case the expression for tuples of the view V-FREE only produces tuples with employees not associated with projects, and fire, being an operation of V-FREE, can only delete tuples that belong to it (in general: tuples in base relations but not in the derived view cannot be modified or deleted by operations defined in the view [1]).

In any case, our approach prevents the execution of operations that violate constraints, rather than allowing any operation to take place and checking afterwards and then undoing the effect of any illegal operation if necessary.

The interdependence of the operations poses certain problems: alterations in one operation may call for a redesign of other operations, even in views of other users. However, in general, it is the expressions section that will be thus affected rather than the displays section; also the application programs using the operations would not be impacted.

An alteration may consist of an additional constraint. For example, we may require that employees be trained on useful skills only; for this we would add the condition that any skill k in acquire should be present in some tuple of REQ. Adding or dropping constraints is largely a policy decision in the enterprise: as an argument against this alteration someone might note that certain skills would be useful even if not explicitly required for some task, or that the training manager should have the flexibility of anticipating the future need for certain skills.

What if the operation suspend a project is eliminated, but the terminate project operation is kept? This alteration implies that the disassociation of employees from the project, which was a side-effect of suspend, can no longer be assumed by (and in fact must now be executed in) terminate.

If the structure of base relations is altered it is still possible in some cases to alter only the expressions (for tuples

and operations) in the views. If relation EMP were decomposed into its projections EMP1 (N,S) ans EMP2 (N,J) easy adaptations in several expressions would be required.

It is also possible to alter the displays themselves. We saw that operations may be created or dropped, but also the specified arguments of existing operations may be changed or the attributes integrating the tuples of a view may be changed. For example, V-TEAM-p mignt be extended with attribute S (salary), taken from base relation EMP, and the operation raise salary by i might be included; the operation would add i dollars to the salary of the indicated employee, with the condition that i be positive (because of the intended effect of the operation, noting that this would also preserve the constraint about the minimum wage).

5. QUERIES ON UPDATES

The design of the views should of course be thoroughly documented. If they are written in a style similar to the one we used in the previous section, the first version (to be refined later towards an implementation) should be kept as part of the documentation.

Much descriptive information can be added to a view as comments, explaining the "meaning" of the view and of each operation, giving indications about what data should be locked during each operation (note that the view construct conveniently mentions in one place the data involved directly and indirectly in an operation), etc.

It is appropriate to keep such descriptive information in the data directory [4]. In this section we shall concentrate on the descriptive (textual) information about update operations which appears in the views and is transferred to the data directory. From there it is made available to the users of the views who are both interested in and authorized to learn about the conditions and side effects of their updates.

Accordingly, we extend the view framework, giving for each condition its meaning and listing other operations that can affect the condition in a negative or positive way together with the authorized user. Similarly, for each side effect its meaning and the operations whose conditions are affected by its execution are given. In both cases under the heading "method" goes the executable part, already seen. Some examples follow:

```
view  V-ALLOC-p
      ..............
      - assign
        conditions . meaning: 'employee must
                       have skills required
                       for task'
                     affected by: 'require-
                       engineering manager'
                       'remove-engineering
                       manager'
                       'acquire-training
                       manager'
                       'lose-training manager'
                     method: for every k in
                       tuples (t,k) in REQ
                       there must be (n,k)
                       in CAP
                     meaning: 'employee must
                       be associated with
                       the project'
                     affected by: 'associate-
                       engineering manager'
                       'disassociate-engi-
                       neering manager'
                     method: there must be
                       some (n,-,p) in ASN
      ..............
end   V-ALLOC-p
```

```
view  V-PROJ
      ..............
      - suspend
        conditions.....
        effects.....
        side effects . meaning: 'employees in
                         project are disasso-
                         ciated'
                       affects: 'restart-en-
                         gineering manager'
                         'terminate-engineer-
                         ing manager'
                         'fire-personnel
                         manager'
                       method: delete (-,-,p)
                         from ASN
      ..............
end   V-PROJ
```

It is up to the designer to include or exclude a given piece of information. Note that, to be rigorous, the operation suspend also affects the first condition for the assign operation, because it disassociates an employee from a project (and when this happens the employee can no longer be assigned to tasks in the project); but suspend also erases the project leader, who is the only authorized user for the assign operation - thus making all conditions vacuous because nobody can open the particular V-ALLOC-p view.

Our examples do not contain texts giving "meaning" and"affects" for the "effects" part of the operations, but it would certainly make sense to include such texts.

By storing textual information in the data directory we create the possibility of automating the retrieval of such information, i.e. of posing queries on updates. This is in fact a different kind of query than the interrogates, because it accesses the data directory instead of the data; it may be the case that a user is authorized, for example, to learn that suspend causes employees to be disassociated from projects without learning however which projects are involved (at each execution of the operation).

Let us consider some useful kinds of queries. Although we again use natural language here, the actual language to be used for the queries could be a trivial, parametric one. The reader will readily locate the texts in the extended views given above that can directly be used for answering each query.

(a) What are the conditions for the assign operation?

(b) What operations can affect the first condition of assign?

(c) What are the side effects of the fire operation?

(d) What operations are affected by the first (in this case only) side effect of suspend?

Another kind of query assumes that, upon the failure to execute an operation, appropriate flags are set indicating which condition(s) failed, and allowing the query to find the appropriate entry in the data directory in order to print it out as answer :

(e) Which condition(s) for operation assign failed?

The "efficient" strategy is to discontinue the execution as soon as the first condition tried fails. An option causing the other conditions to be also tested might be useful.

One can think of sequences of queries posed by a user. A common situation would be query (e) followed by (b): the project leader

fails when attempting to assign an employee to a task, finds out that the first condition (lack of some skill) is the problem, discovers or is reminded of the existence of remove and acquire - which are alternative ways to remedy the situation - and communicates with one or the other of the authorized users asking for their action. Regular interrogates on the data are issued (by whoever is authorized) to find out which are the missing skills.

Communications among the users are an important feature, as we should expect from the interdependence of views. Good communications can enhance the necessary user cooperation.

Many other communications are brought to mind in our simple example. The engineering manager through query (d) finds out that by disassociating an employee from a project he may be making it possible to fire the employee (if it was the last project with which the employee was associated). He may want to ask the personnel manager not to perform the fire operation, because perhaps he intends to use the employee shortly in some other project. All such communications can of course be done through the terminals of the users involved, and bring up no new problems.

A more ambitious strategy is for a user to request that he be automatically notified when another user performs a certain operation with indicated arguments. Or he may schedule his operation to be executed immediately after the other user's operation, as for example:

assign(n,t) after acquire (n,k)

if k is the only missing skill for n to be assigned to t. We must note that this does not guarantee that assign will be successful, bacause in an environment where concurrent updates occur it is still necessary to test for all conditions when it is time to execute assign and maybe a condition that held before no longer holds.

Alternatively we could schedule the operation to be executed as soon as all conditions hold, whatever they are:

assign (n,t) when possible

Such automatic actions do require special features, whose cost should be compared with the benefits of avoiding delays between the actions and the personal exchange of communications. Implementations may use tools proposed for scheduled routines [16] and persistent tasks [17]. Human factors studies are needed to assess how users would react to this environment.

Finally, as the interactions among update operations grow in complexity, it may be useful to provide for the simulated execution of sequences of operations. The simulation would take place on snapshosts, i.e. views "computed" from their expressions and stored as ordinary relations. One obvious drawback is that once computed a snapshot no longer corresponds to the views, unless these are not updated until the end of the simulation run.

6. CONCLUSIONS

According to our approach, at any given time, there exist in a data base a fixed number of views with specified operations defined on them, and all the views have been designed by and therefore are known to a group of administrators. It is possible to create, change, or drop views, but the process in-

volved is troublesome because the impact on other views must be carefully considered and may require additional adaptations; the same group of administrators is in charge of this revision process.

In such an environment it is possible to determine what configurations can or can not arise in the data base, as a result of sequences of the existing update operations. As the behavior of a data base becomes more predictable, the data base is no longer an object of unmanageable complexity [18], and opportunities are created for implementation decisions leading to improved efficiency, as indicated with strategies for enforcing constraints.

The disadvantages of our approach are generally related to a certain loss in flexibility. Also, the proof that conditions, effects and side effects spread over possibly several views guarantee that a constraint is preserved may be difficult and will often be long and tedious.

However the approach appears to be a realistic one for data bases with not too numerous or too complex constraints and with relatively few kinds of update operations, and where, in addition, the required update operations do not have to be altered very frequently. Our experience indicates that a data base can be quite large in terms of bytes of storage without being complex in terms of these criteria. It is also realistic to expect that more transaction-like operations [1,14] would seem more appealing to business-oriented users.

The main lines of this research have been initially sketched in [19] and further developed in [3,8]. More work is needed, for example, for coping with deeper semantic problems arising in data bases not as simple as the one described here, and where the notion of a hierarchy of abstractions [20] is of help; for accommodating this notion we would extend our approach to allow views defined in terms of other views (rather than base relations), which parallels the provision of external schemas defined in terms of other external schemas made in [4].

REFERENCES

1. Date, C.J., "An introdution to database systems", second ed. - Addison-Wesley (1977).

2. Brodie, M.L., "A formal approach to the specification and verification of semantic integrity in data bases", Ph.D. Thesis, University of Toronto, in preparation.

3. Furtado, A.L. and Sevcik, K.C., "Permitting updates through views of data bases", T.R. 2/78 - PUC/RJ (1978).

4. Tsichritzis, D. and Klug, A., "The ANSI/X3/SPARC DBMS framework", T.R. 12, University of Toronto/CSRG (1977).

5. Chamberlin, D. et al., "Sequel 2: a unified approach to data definition, manipulation and control", T.R. 1978 ## 26096, IBM Research (1976).

6. Hoare, C.A.R. and Wirth, N., "An axiomatic definition of the programming language Pascal", Acta Informatica, vol. 2,4 (1973) 335-355.

7. Paolini, P. and Pelagatti, G., "Formal definitions of mappings in a data base", Proc. SIGMOD Conference (1977).

8. Sevcik, K. Furtado, A.L., "Complete and

compatible sets of update operations", T.R. 26/77, PUC/RJ (1977).

9. Liskov, B. et al., "Abstraction mechanisms in CLU", CACM 20, 8 (1977) 564-576.
10. Shaw, M. et al., "Abstraction and verification in Alphard: defining and specifying iteration and generators", CACM 20,8 (1977) 553-564.
11. Geschke, C.M. et al., "Early experience with Mesa", CACM 20,8 (1977) 540-553.
12. Popek, G.J. et al., "Notes on the design of Euclid", Proc. of ACM Conference on Language Design for Reliable Software" (1977).
13. Parnas, D.L., "A technique for software module specification with examples", CACM 15,5 (1972) 330-336.
14. Schmidt, J.W., "Some high-level constructs for data of type relation", Proc. of SIGMOD Conference (1977).
15. Jones, A.K. and Liskov, B.H., "A language extension for controlling access to shared data", IEEE Transactions on Software Engineering, SE-2, 4 (1976) 277-285.
16. Pratt, T.W., "Programming languages: design and implementation", Prentice-Hall (1975).
17. Sutherland, W.R., "Distributed Computation Research at BBN", vol. III BBN T.R. 2976 (1974).
18. Hoare, C.A.R., "Data reliability", Proc. of International Conference on Reliable Software", (1975).
19. Furtado, A.L. and Lucena, C.J., "An incremental approach to the construction of data base software", Prof. of AICA Conference, Pisa (1977).
20. Smith, J.M. and Smith, D.C.P., "Database Abstractions": Aggregation and Generalization", TODS 2,2 (1977) 105-133.