AN INFORMAL APPROACH TO FORMAL SPECIFICATIONS

A.L. Furtado

Pontifícia Universidade Católica do R.J.

Brasil

ABSTRACT

An attempt is made to help bridging the gap between theoretical research and the real practice of data base specification. Some fundamental problems that must be addressed are described and a number of formalisms that propose to attack them are overviewed. The case pro and against the use of formal specifications is presented.

Keywords: data bases; formal specifications; integrity constraints; data models; logic; abstract data types; denotational methods; grammars.

1. Introduction

Formal specifications have been used in several areas of computer science, particularly in programming languages and in data bases. Efforts in the former area started first and have established the main lines for subsequent work (see [Pag], for example). The present paper will concentrate on the latter area.

More specifically, we shall be concerned with the use of formal specifications in the design of data base applications, i.e. particular data bases for some application area of an institution.

During the design process a number of questions must be answered, namely:

a. What information will be needed?
b. How is the information going to be used?
c. Under what form will the information be kept?

The aspects underlying these questions are, respectively: information, operations and representation. The answers should be satisfactory, in the sense that the information will be meaningful, valid and useful to the activities of the institution; the information should also be easily and efficiently accessible.

Our first tendency would be to attack the problem with the usual ad-hoc methods. However:

> Dijkstra: "The problems of business administration in general and data bases in particular are much too difficult for people that think in [programmerese], compounded with sloppy English." [Di].

So, we should resort to some formal methodology, but is the problem tractable through the existing formalisms? Some people feel otherwise:

> Hoare: "... the recommendation to remove references ... runs counter to the still prevalent belief in integrated information systems, relational data bases, etc. and suggests that it may be preferable to go back to earlier, simpler techniques using separate files without cross references ..." [Hoa].

The above remark goes beyond a negative assessment of the current formalisms for a problem of such complexity. It suggests that we should drop the problem as unfeasible. Should we just give up? We cannot afford to do so because there is a real need clearly manifested by the business world:

> Sibley: "Yet there are substantial problems and industry has started to solve them. It would be terribly sad if we manage to prove once again that 'computer science' is irrelevant to the real practice of computing." [Si].

After all these pessimistic remarks, one would expect at least that some constructive effort is being done in the many specialized data base journals and conferences. Such work is indeed going on, but as to its impact on the community:

> Gotlieb: "The lesson about data bases to be drawn from this is that there is a real danger that the research carried out in universities and industrial laboratories, as described in VLDB conferences, will have very little effect on the large operational data bases used by the world at large, and on which most money is spent." [Go].

With respect to the more formal research the main communications barrier does not seem to result from any difficulty with the formalisms themselves, but from the unawareness of practitioners of what researchers propose to do, and perhaps from a certain unwillingness of the latter to divulge their goals in less than rigorous terminology. In this paper we shall try to understand the problem and provide an indication of how formal approaches propose to attack it.

Most people take the ANSI/X3/SPARC architecture

[TK] as a common starting point. To follow this architecture, one divides the design process into three phases, wherein three schema levels are treated; these are the conceptual schema (a high level description of the entire data base), the external schemas (partial high level descriptions, of interest to different users) and the internal schema (describing the implemented data base). Our discussion will concentrate on specifications at the conceptual schema level.

Various difficulties are encountered in the specification of conceptual schemas, starting with the lack of a standard terminology. A recent report has done some contribution in this regard [Gr]. We feel that both the terminology and the approach make communications difficult, because of the background of the people involved. The application area specialists have their own jargon and of course cannot be expected to communicate in any formal or ad-hoc computer science jargon. 'Neutral' information systems terminology has been tried but even that can be unsatisfactory in many cases. It appears that the designers, regardless of their background, are forced to learn the specific requirements of an application area as expressed in the application area terminology; this has to be done at the crucial first stage (informal description, in fig. 1 below) where the prospective users are led to communicate their needs and expectations. In other words, one must learn first how that particular business works, so as to be in a position to assess what is important and what is accessory.

Characteristics of different application areas can vary dramatically and widely different kinds of data can be involved. Compare, for instance, data bases of the traditional "parts and suppliers" style with data bases for census data, airline reservation, geographical information, satellite observations, business forms, texts, etc.

The response to all this variability given by the current data models and methodologies and, even worse, by the commercially available data base management systems (DBMSs) is far from satisfactory.

In sections 2, 3 and 4 we shall give an informal view of what we consider to be the main goals of formal approaches, using for this purpose the three basic aspects that we detected at the outset: information, operations and representation (fig.1). The discussion will be illustrated by a simple example.

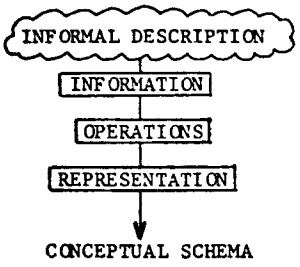

Fig.1: The three levels of specification

Section 5 overviews the research work on formalisms. Section 6 presents the case pro and against data base specifications.

2. First level: information contents

At the first level, we say that data bases will contain instances of facts, defined as positive assertions within the application area. Usually, negative facts are not stored (such as what courses are not taken by a trainee). A state is the collection of facts that are true at a given instant of time; therefore, a state denotes the entire contents of the data base at that instant. Static constraints are restrictions defined on states. A valid state is one that conforms to all specified static constraints.

A transition is a state transformation. Transitions can be denoted by pairs (or, more generally, sequences) of states. Again we may want to impose restrictions on transitions. Thus a valid transition, besides being required to involve only valid states, must conform to the declared transition constraints.

We now introduce the simple example to be used throughout the discussion. The example belongs to the area of personnel training of an enterprise, and the data to be maintained refers to one training term. Using the terminology of the present level we have:

FACTS: - courses are offered
- trainees take courses

STATIC CONSTRAINT: trainees can only be taking currently offered courses

EXAMPLE OF A VALID STATE: c1 is offered
c2 is offered
John takes c1

TRANSITION CONSTRAINT: the number of courses taken by a trainee cannot drop to zero (during the training term)

EXAMPLE OF A VALID TRANSITION:

c1 is offered		c1 is offered
c2 is offered	->	c2 is offered
John takes c1		John takes c2

Certain points not explicit in our unrealistically simple example must now be stressed. The first point is time [BADW], implicitly involved in the transition constraint, which prevents the number of courses taken by a trainee to drop to zero only within the training term under consideration; without this proviso the constraint would not make sense. Time appears in many ways: as simply a way to order the states, as a duration, as a date, etc. Next we observe that the size and complexity of realistic data bases make their direct specification in one piece an impracticable task. Modularization is in order. Also, we must have ways to verify properties of specifications, such as their consistency, non-redundancy, etc., both within and across the various modules.

In an "intelligent" data base, one should be able to infer certain facts, which then would not have to be stored. For example, if we know that the cur-

rent state of our training data base is valid and that John takes c1, we could deduce that c1 is offered, taking our static integrity constraint as a general law. Inference becomes more complex when states are allowed to include alternative facts such as John takes c1 or c2, or facts with indeterminate values, such as John takes some course whose name is presently unknown (a so-called null value).

The relevance of the first level of specification had an early recognition with the infological approach [Su].

3. Second level: operations

In view of the first level characterization in terms of facts, we have an obvious choice for the primitive operations that will achieve state transitions:

- create
- assert <fact>
- deny <fact>

the first operation being needed for initializing the data base to the initial "empty" state, where no facts are true. We also need an operation to check whether a fact is currently true:

- <fact>?

The execution of certain sequences of assert and deny operations may have the property that the original and the final states are valid as well as the transition between them, even though some intermediate states or transitions perhaps violate some constraint. Such sequences will be called transactions. For different classes of states a sequence may or may not have the transaction property. The favorable case will be characterized by a suitable pre-condition, which is a logical expression, possibly involving the inspection of certain facts in the current state. We now define application operations as operations that:
- correspond to actions happening in the application world that may cause certain facts to hold or cease to hold

and have the general format: if pre-condition then transaction (else do nothing).

In our example the application operations will be:

initiate training term: create
offer course: assert course is offered
cancel course: if no trainee takes the course
then deny course is offered
enroll trainee in course: if the course is offered
then assert trainee takes course
transfer trainee from course 1 to course 2:
if the trainee takes course1
and does not take course2
and course2 is offered
then deny trainee takes course1
assert trainee takes course2

We are now in a position to introduce the encapsulation design goal, coming from the abstract data type approach: if only (correctly specified) application operations are ever used, all static and transition constraints will be preserved. Encapsulation implies that primitive operations cannot be used directly. This, of course, reduces the freedom of users to handle kinds of updates whose need was not anticipated; one should decide in each case whether the loss of freedom is a reasonable price to pay for the integrity of the data base. Also remark that encapsulation need not restrict the use of queries (<fact>?) [Zi]. Queries can never be harmful to integrity, although we may want to restrict their usage for authorization considerations.

When we associate application operations with actions happening in the real world, we are assuming that reality is changed only when the corresponding application operations succeed in updating the data base in the intended way. In other words, the segment of interest of the real world is undistinguishable from the data base. It becomes physically impossible to perform an action that violates some policy of the institution, expressed as a constraint.

We must recognize that this assumption is not always realistic. There may be actions performed outside the institution having results that we want to record in the data base (ex: the local minimum wage is changed, a new tax is created). Sometimes the institution may decide to act contrarily to its policies, thereby deliberately disregarding self-imposed constraints (ex: granting unusual discounts), and in this case too we merely record the fact. In both cases, the data base would be incorrect if such facts were not recorded, in the fundamental sense of a disagreement between reality and the data base.

In what follows we shall concentrate on the simple situation where things happen through the data base so that we may ignore the problems above. Also, wherever we assume that the real world and the data base coincide, we can automate certain actions, achieving the so-called active systems, where actions can be triggered by the occurrence of certain events (logical expressions, possibly involving stored facts, similar therefore to pre-conditions).

There may be more than one way to design application operations that effectively preserve the declared constraints. Part of this freedom of choice comes from the existence of different ways to combine pre-conditions and primitive operations and also to alternate actions initiated by users with triggered actions. To discuss these possibilities we shall employ the example in fig. 2.

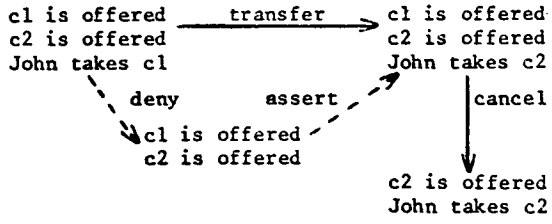

Fig. 2: using the application operations

With the present definition of the operations we could not execute cancel(c1) at the state where John takes this course, whereas the use of cancel is legal at the state reached by transferring John to c2 (note, incidentally, the decomposition of

transfer into primitive operations with the occurrence of a transition that would be illegal by itself, since at the intermediate state John takes zero courses). This suggests that, in order to make cancel applicable at the first state shown, we might redefine cancel by expanding its corresponding sequence of primitive operations thereby being able to weaken its pre-conditions. The new sequence of assert and deny operations would place all trainees taking c1 alone in some other course (if such exists) and would simply remove any other trainees from c1.

Besides redefining an operation, we are free to create additional operations. For instance, we might add an operation allowing a trainee to drop a course, if it is not the only one that he is currently taking. Finally, if we had both drop and transfer operations, we could achieve the modified effect of cancel indicated above without redefining the operation: we should merely add a trigger causing either drop or transfer to be invoked for each trainee taking c1.

This shows that, because more than one second level specification may be compatible with the same first level specification, we may substitute a second level version for another while the first level remains stable. The first level does not entirely determine the second, since defining the repertoire of operations is a design decision to be made at the second level, often with considerable flexibility.

The order of execution of operations, on the other hand, is not entirely free because of the interplay of pre-conditions and effects. In our example, enroll(John,c1) can only be executed after offer(c1) has been executed. Operations whose effects are necessary to fulfill pre-conditions of other operations entail serial execution. On the contrary, offer(c1) and offer(c2), for instance, can be executed in parallel.

Also from the study of the interplay of pre-conditions and effects, one can conclude that different sequences of operations can accomplish the same net result. For instance, starting at the initial empty state, the two sequences:

offer(c1); offer(c2); enroll(John,c1);
transfer(John,c1,c2); cancel(c1)

and

offer(c2); enroll(John,c2)

lead to the same state and are in this sense equivalent.

4. Third level: representation

One of the ways to specify a data type is to employ "some established mathematical discipline ... used to provide a high level abstract implementation or model of the desired data abstraction" [LZ]. We claim that this describes accurately the main purpose of data models. The mathematical disciplines of the current data models involve trees, graphs, tables, etc. as abstract representations.

Most formal work on data bases aims at formalizing some data model, rather than a specific data base application. The assumption is that data models have not been presented in a sufficiently formal way, even the more mathematically-oriented ones such as the relational model.

If we regard a data base application as a data type on its own right and a data model as another (less restricted) data type, then it is natural to view the process of going from the second to the third level of specification as the implementation of a data type by another [GHM]. The process consists of finding a representation for data base application states in terms of the (tree, graph, table, etc.) structure underlying the data model and of writing "programs" for each application operation using the data model operations.

Figure 3 shows how our example would be represented according to the entity-relationship model [Ch]. Trainees and courses are entities, trainees being "weak" entities since they only exist in the data base while taking some course. The presence of a course means that it is being offered. A course may be involved in at least zero and at most an arbitrary number of occurrences of the takes relationship (i.e. each course may be taken by zero or more trainees), whilst the lower and upper limits for trainees are 1 and an arbitrary number, respectively.

Fig. 3: representation in the ER model

The static constraint that a trainee can only take courses that are being offered is implicit in this representation, because it follows from the graph constraint that an edge (relationship) can only exist between existing nodes (entities). The transition constraint that the number of courses taken cannot drop to zero is only partly implicit, however. We have only managed to express that while a trainee is present in the data base he must be taking at least one course, but there is no convention in ER-diagrams to express that once inserted a trainee cannot be removed. Whatever is not implicit, or can somehow be expressed in a declarative way provided by some feature of the data model, should be expressed procedurally within the "programs" implementing the application operations.

Usually the entity-relationship model is only the first step in the representation level. Some other data model corresponding to commercially available Data Base Management Systems (DBMSs) must be used if one wants to pass eventually from the conceptual to the internal schema, where a computer implementation will be achieved. Since from [Ch] strategies have been proposed for obtaining representations in the relational, network or hierarchical models [Da] from an entity-relationship representation.

Here we stress that a thorough knowledge of the data base application resulting from levels 1 and 2 is fundamental for the choice of an adequate representation, both the first (entity-relationship) and the derived ones. The "automatic" solution for

deriving a relational representation would lead to two "entity-relations" (for trainees and for courses) and one "relationship-relation" (for the takes relationship). Yet the constraints identified at level 1 and the definition of operations at level 2 suggest that the "entity-relation" for trainees is unnecessary. Perhaps two relations COURSES(COURSE NAME,NUMBER OF ENROLLMENTS) and TAKES(TRAINEE, COURSE NAME) would constitute a reasonable way to structure the data base.

Besides diagrams, data models usually provide languages to express the structure and manipulation of data bases. Given the wide variability of the features of data base applications, mentioned in the introduction, it seems convenient to adapt the languages to accomodate such differences. One might characterize the language of a data model as an extensible language, there being a "customized" language for each application (or family of applications).

Data models have been extended [TL], mainly by incorporating ideas coming from research in artificial intelligence. These ideas include semantic hierarchies (e.g. is-a and part-of, corresponding to the generalization and aggregation data abstractions) [SS], semantic networks [Fi] and frames [Mins,Rou].

We remarked, in the previous section, that passing from the first to the second level involves choosing among possibly many compatible alternatives; the same is true about the passage to the third level: different data models can be chosen. Hence, higher levels are more stable and remain unaffected when changes are introduced in the lower levels, such as adding new operations, at the second level, or shifting to a different data model or performing a restructuring within the same data model, at the third level.

5. A flurry of formalisms

Some of the techniques that have been used in the formal specification of data bases are:

a. first order logic [GM, GMN, Ga, Ja];
b. special logics, particularly dynamic logic [CB], temporal logic [CF], modal logic [Li] and non-monotonic logic [Mink].
c. algebraic presentations [EKW, LMWW, BZ, VCF];
d. denotational methods, with a special emphasis on the Vienna Development Method (VDM) [NO, BL];
e. grammars based on strings [RB] or on graphs [Ful, EK, FV2] and two-level grammars [Fu2].

The use of logic leads naturally to the investigation of consistency, non-redundancy and other properties of specifications [CCF]. It also leads to various applications of inference, particularly by looking at data bases as collections of facts extended with general laws. Another use of inference lies in automatic synthesis, where a plan is determined with the help of some theorem-proving algorithm, for traversing the state space. The goal is to achieve a transition from the current state to some state where the desired facts are true, both the transition and the state reached being valid [SMF]. Finally, systems based on theorem-proving, e.g. Prolog [WPP], can run specifications expressed in logic. Running specifications allow testing and experimentation (prototyping); the importance of this feature comes from the impossibility to prove the equivalence between an initial informal description and (any kind of) formal specification.

Dynamic logic has been used to construct paradigms for data base languages, specifying the various commands within an axiom system in order to permit formal proofs about programs consisting of such commands. Temporal logics aim at the formalization of complex transition constraints; they can also cope with the description of active systems, including events and triggers. Both formalisms are appropriate to investigate the synchronization problems related to parallel or serial execution of operations. Modal and non-monotonic logic have been applied to indefinite data bases, which are those containing alternative or indeterminate facts.

Algebraic presentations take the abstract data type approach. A state is referred to by some sequence of application-oriented operations (starting at the initial empty state) able to generate it. Such sequences have been called traces [BP,VF], which recalls that they contain a "history" of how the state might be reached. The fundamental role is played by (possibly conditional) equations, allowing to determine whether two or more traces lead to the same state; among such equivalent traces one may choose, according to appropriate criteria, a canonical trace (canonical term, in [GTW]) to designate the state uniquely. As happens with specifications based on logic, algebraic specifications are amenable to prototyping [Ge,FV1, BGW]; operations can be executed symbolically as transformations on traces [FVC]. A formal treatment of modularization, where modules are data types wherefrom other data types are constructed, has been proposed in [BG] and applied to data bases [DMW,SFNC].

The fundamental idea of denotational specifications is to explain each construct in mathematical terms [Sco,Te]. This makes them ideal for the formalization of data models, data languages and entire DBMSs. A standardization effort [BS] has included a VDM specification of the relational model, formalizing the structural aspects and the syntax and semantics of operations; constraints related to the structure and to the operations have been duly considered. One would expect that similar techniques could be used to formalize data base applications also in mathematical terms; next, one could verify if a description of the data base application using the data model is faithful, i.e. mathematically equivalent to the formalization.

Grammatical formalisms are directed to specify the syntax of some language. Graph-grammars are useful whenever we want to concentrate on the structural connections existing among data base components and/or to describe intuition-appealing languages based on diagrams. Two-level grammars [Wi] are particularly fit for handling "context-sensitive" (i.e. non-local) constraints, thus facilitating modular design; they are even powerful enough to formalize the semantics of operations. By separat-

ing the rules of a two-level grammar into general and specific, we can specify both a data model and each application, thereby effectively converting the language of the data model into an extensible language.

It has been widely recognized [Do,HL,Pag,CPLM,VCF] that different formalisms serve better different purposes, so that it may be convenient to provide two or more complementary specifications to formalize the various aspects of the same data base application.

6. An "abstract" dialog

Author: Although much remains to be done, I claim that some practical results have emerged from the above research efforts. We can already envisage the complete formal description of a data-based information system. Having the complete description, one would then decide which rules would be automated and which would remain as (well-documented) precepts to guide the human agents, externally therefore to the data base.

Practitioner: There is a peculiar aspect in your presentation. To describe your example data base you did not use any formal notation. Should I conclude that, instead of insisting on formalisms, you should simply advocate a rigorous usage of natural language?

A: Unfortunately this is not enough in many cases [Par]. Let me show you a rather surprising example of the ambiguities inherent in natural language: the famous transition constraint that "salaries cannot decrease".

P: Is that ambiguous? Even the way to enforce it seems quite obvious. You just compare the old and the new salary upon each insertion or modification of the salary item [Da].

A: The ambiguity does not reside on how you enforce the constraint but on its very meaning, which, expressed in temporal logic [CF], can be, among other alternatives:

(a) $\neg\exists n\exists s(\Diamond(EMP(n,s);\exists s'(EMP(n,s')\wedge s>s')))$
(b) $\neg\exists n\exists s(\Diamond(EMP(n,s)\wedge\Diamond(\exists s'(EMP(n,s')\wedge s>s'))))$

Expression (a) reads:

(a')"it is false that there is an employee n and salaries s and s' such that eventually n has salary s in one state and salary s', less than s, in the next state".

Expression (b) looks almost the same, except that "in the next state" is replaced by "in some future state". This little distinction is crucial: expression (a) does not prevent you from firing an employee and hiring him again with a lower salary, while (b) says that you cannot do that.

P: Good grief! Would you require a user to analyse these abstruse expressions?

A: No. The scenario that formal specifiers gifted with common sense would have in mind is:

- a user transmits to the specifier the constraint "salaries cannot decrease";

- the specifier writes the constraint in formal notation;

- since an adequate formalism forces the specifier to be precise about the state with salary s' being either the next state or some future state, he goes back to the user and asks the question.

There is no need to show the users the formal expressions. It is however a good practice to show them a "translation" of the expressions into natural language, like (a'). These translations tend to be longer than the original formulations made by the users, and they are rightly so because, as demonstrated, they can bring to the fore troublesome points that might otherwise be ignored.

P: It is a matter of conjecture to determine whether the occurrence of ambiguities is frequent enough to justify formalisms. Some people feel that the effort to develop a formal specification does not pay-off. Well-written comments, in natural language of course, are almost always easier to understand [Row].

A: But comments are written together with programs and their sole purpose in documentation, whilst specifications propose to determine, beforehand, what programs will be supposed to do. A specification does pay-off, even more, when it is executable so that one can experiment with it, perhaps changing it several times as demanded by the future users, before commiting oneself to a lengthy and costly implementation. Furthermore, it is useful to have this specification cast in a style that favors rigorous verifications of correctness. Finally, the executable specification can be re-activated during the maintenance phase, in order to experiment with changes necessitated by new needs of the user community.

P: It is certainly nice to have an executable specification available with all those features. Incidentally, the idea of producing a first version for experimentation only has been defended by software engineers [Broo] and has been used with good results reported (as in [As], for example). However, I have certain misgivings as to the effort needed to produce it, particularly when formal methodologies are employed. First, let me point out the matter of scale. You were discussing a very small example. I fear that producing a specification for any realistic data base application and making it reliably consistent would be a difficult job indeed, even when modular strategies are used.

A: There are examples of non-trivial applications being specified using formal techniques [GH,Sch], and there are cases [BP] where they helped finding, not only ambiguities, but even errors that several people had failed to detect in a preliminary informal specification, errors that would be tricky and expensive to correct in the programming phase.

P: Yes, but the specifications were done by academic people. Also you need programming languages based on logic or on symbol-manipulation, whereas most professional programmers are not trained to use them. The size itself of the programming task in the case of realistic applications could be such that, for having the executable specification, we would perhaps double the time and cost of embarking on an implementation after a simpler requirements

analysis.

A: Designing a data base application is decidedly not an outright programming job, unless one confuses it with the mere superimposition of a DBMS over a number of existing files. Highly trained people are needed, although , admitedly, not necessarily university people... Anyway, it is in order to simplify this task and also to take care of the bulk of the effort involved that software tools have begun to appear (e.g. [Ge,GT]).

P: How do I communicate my intentions to software tools that take the algebraic approach? Can I just indicate the queries and updates with their preconditions and effects?

A: In general you are required to supply the equations that show what sequences of application operations are equivalent. But it may be possible to construct some interface enabling you to communicate through it as you said and having the system (perhaps with your help, interactively) derive the equations.

P: I look forward to seeing such software tools widely available and featuring user-friendly interfaces. It is fine to praise formal "non-procedural" methodologies for their freedom from programming and other implementation details; but to dispense the users of the methodologies from knowing about programming by demanding in exchange that they become logicians or mathematicians is totally unrealistic. Another objection that I have to the formal specifications that we have been discussing is that they leave out more than implementation details. Certain properties of systems such as speed and space requirements, memory access patterns, reliability [Sh], concurrency, security, recovery, exception handling [Brod], etc. are left out as well or, at best, insufficiently treated.

A: This may still be partly true. Yet some of the aspects that you enumerated belong to the internal schema rather than to conceptual schema design. The main benefits of executable specifications, at the current state of the art, refer to testing the behaviour of a data base application subjected to integrity constraints, thereby giving the prospective users an opportunity to assess it and react to it. Executable specifications can help in preparing for the phase when the requirements that you mentioned will be considered, if you put them to a monitored usage and collect some statistics.

P: Other important properties, related to users' views, are also missing. In most cases you cannot first design the conceptual schema and then divide it among the users; very seldom you will find in an institution a person or group of persons with a global knowledge of the application area, able to give you a complete description. The usual bottom-up strategy is to extract from the various prospective users their specialized views and then integrate the views [NG], identifying and disciplining the interferences among them.

A: Views belong to external schema design, and the paper deals with the conceptual schema only, as promised in the introduction... However it is fair to indicate here that some theoretical work about external schemas and the interference problem is under way [Pao,CCF]. About view integration, the proposed usage of the data abstractions introduced by [SS] is worth mentioning [TF].

P: I submit that we still have to wait for reports on actual usage of all this by people working in the business environment, as has been done for structured programming and top-down design (in [Ho], for example). Only from the analysis of such reports one will be able to settle the case of whether this line of research is relevant to practitioners, or will remain exclusively as a contribution towards the understanding of data and data-handling functions.

References

[As] M.M. Astrahan et al - "A history and evaluation of System R" - Research report RJ2843(36129), IBM S.Jose (1980).

[BADW] A.Bolour, L.Anderson, L.Dekeyser and H.Wong "The role of time on information processing" SIGMOD Record, 12, 3 (1982) 27-50.

[BG] R.W. Burstall and J.Goguen - "Putting theories together to make specifications" - Proc. Fifth International Joint Conference on Artificial Intelligence (1977) 1045-1058.

[BGW] R.M. Balzer, N.M. Goldman and D.S. Wile - "Operational specification for rapid prototyping" - Technical Report, Information Sciences Institute, USC (1981).

[BL] D. Bjorner and H.H. Lovengreen - "Formalization of database systems and a formal definition of IMS" - Proc. 8th International Conference on Very Large Data Bases (1982) 334-347.

[BP] W.Bartussek and D.Parnas - "Using traces to write abstract specifications for software modules" - UNC Report 77-012 - University of North Carolina at Chapel Hill (1977).

[Brod] M.L.Brodie - "Type specification and databases" - topic outline for a panel in 8th International Conference on Very Large Data Bases (1982).

[Broo] F.P. Brooks - "The mythical man-month" - Addison-Wesley (1979).

[BS] M. Brodie and J. Schmidt (eds) - "Final report of the ANSI/X3/SPARC DBS-SG Relational Database Task Group - SIGMOD Record, 12, 4 (1982).

[BZ] M.L. Brodie and S.N. Zilles (eds.) - Proc. of the Workshop on Data Abstraction, Databases and Conceptual Modelling - SIGMOD Record, 11, 2 (1981).

[CB] M.A. Casanova and P.A. Bernstein - "A formal system for reasoning about programs accessing a relational database" - ACM TOPLAS 2,3 (1980) 386-414.

[CCF] M.A. Casanova, J.M.V. de Castilho and A.L. Furtado - "Properties of conceptual and external schemas" - Proc. Formalization of Programming Concepts - North-Holland (1982) to appear.

[CF] M.A. Casanova and A.L. Furtado - "A family of temporal languages for the description of transition constraints" - Proc. Workshop on Logical Bases for Databases (1982) to appear.

[Ch] P.P. Chen - "The entity-relationship model - toward a unified view of data" - ACM TODS 1, 1 (1976).

[CPLM] R.L. Carvalho, A. Pereda B., C.J.P. Lucena and T.S.E. Maibaum - "Data specification methods" - Proc. International Conference on Systems Methodology, Washington (1982).

[Da] C.J. Date - "An introduction to database systems" - Addison-Wesley (1981).

[Di] E.W. Dijkstra - "How to tell truths that might hurt?" - SIGPLAN Notices, 17, 5 (1982) 13-15.

[DMW] W. Dosch, G. Mascari and M. Wirsing - "On the algebraic specification of databases" - Proc. 8th International Conference on Very Large Data Bases" (1982) 370-385.

[Do] J.E. Donahue - "Complementary definitions of programming languages semantics" - Springer (1976).

[EK] H. Ehrig and H.J. Kreowski - "Applications of graph grammar theory to consistency, synchronization and scheduling in data base systems" - Information Systems, vol.5 (1980) 225-238.

[EKW] H. Ehrig, H.J. Kreowski and H.Weber - "Algebraic specification schemes for data base systems" - Proc. 4th International Conference on Very Large Data Bases (1978) 427-440.

[Fi] N.V. Findler (ed.) - "Associative networks: representation and use of knowledge by computers" - Academic Press (1979).

[Ful] A.L. Furtado - "Transformations of data base structures" - in "Graph-grammars and their application to computer science and biology" - V. Claus, H. Ehrig and G. Rozenberg (eds.) - Springer (1979) 224-236.

[Fu2] A.L. Furtado - "A W-grammar approach to data bases" - Monograph 9/82, PUC/RJ (1982).

[FV1] A.L. Furtado and P.A.S. Veloso - "Procedural specifications and implementations for abstract data types" - ACM/SIGPLAN Notices, vol. 16 no. 3 (1981) 53-62.

[FV2] A.L. Furtado and P.A.S. Veloso - "Specification of data bases through rewriting rules" - Proc. 2nd International Workshop on Graph Grammars and their Applications to Computer Science (1982) to appear.

[FVC] A.L. Furtado, P.A.S. Veloso and J.M.V. de Castilho - "Verification and testing of S-ER representations" - in 'Entity relationship approach to information modelling and analysis' - P.P.Chen (ed.) - E-R Institute (1981) 125-149.

[Ga] H. Gallaire - "Impacts of logic on data bases" - Proc. 7th International Conference on Very Large Data Bases (1981) 248-259.

[Ge] S.L. Gerhart et al - "An overview of Affirm: a specification and verification system" - Proc. IFIP (1980) 343-348.

[GH] J. Guttag and J.J. Horning - "Formal specification as a design tool" - Proc. of the 7th Annual Symposium on Principles of Programming Languages (1980) 251-261.

[GHM] J. Guttag, E. Horowitz and D.R. Musser - "The design of data type specifications" in 'Current trends in programming methodologies' - R. T. Yeh (ed.) vol. IV - Prentice-Hall (1978).

[GM] H. Gallaire and J. Minker - "Logic and data bases" - Plenum Press (1978).

[GMN] H. Gallaire, J. Minker and J.M. Nicolas - "Advances in data base theory" - Plenum Press (1981).

[Go] C.C. Gotlieb - "Some large questions about very large data bases" - Proc. Sixth Very Large Data Bases Conference - (1980) 3-7.

[Gr] J.J. van Griethuysen (ed.) - Concepts and terminology for the conceptual schema and the information base" - report from the ISO TC97/SC5/WG3 group (1982).

[GT] J.A. Goguen and J.J. Tardo - "An introduction to OBJ: a language for writing and testing formal algebraic specifications" - Proc. Specifications of Reliable Software - IEEE Computer Society (1979).

[GTW] J.A. Goguen, J.W. Thatcher and E.G. Wagner - "An initial algebra approach to the specification, correctness and implementation of abstract data types" - in 'Current trends in programming methodology' - R.T. Yeh (ed.) - vol IV - Prentice-Hall (1978) 80-149.

[HL] C.A.R. Hoare and P.E. Lauer - "Consistent and complementary formal theories of the semantics of programming languages" - Acta Informatica 3 (1974) 135-153.

[Hoa] C.A.R. Hoare - "Data reliability" - Proc. International Conference on Reliable Software (1975) 528-533.

[Hol] J.B. Holton - "Are the new programming techniques being used?" - Datamation - July (1977) 97-103.

[Ja] B.E. Jacobs - "On database logic" - J. ACM, 29, 2 (1982) 310-332.

[Li] W. Lipski - "On databases with incomplete information" - J. ACM, 28, 1 (1981) 41-70.

[LMWW] P.P. Lockemann, H.C. Mayr, W.H. Weil and W.H. Wohlleber - "Data abstractions for data base systems" - ACM TODS, 4, 1 (1979) 60-75.

[LZ] B. Liskov and S. Zilles - "An introduction to formal specifications of data abstractions" in 'Current trends in programming methodologies' - R.T. Yeh (ed.) - vol I - Prentice-Hall (1977) 1-32.

[Mink] J. Minker - "On indefinite databases and the closed world assumption" - Technical Report 1076, University of Maryland (1981).

[Mins] M. Minsky - "A framework for representing knowledge" - in 'The psychology of computer vision' - P.H. Winston (ed.) - McGraw-Hill (1975) 413-424.

[NG] S.B. Navathe and S.G. Gadgil - "A methodology for view integration in logical data base design" - Proc. 8th International Conference on Very Large Data Bases (1982) 142-155.

[NO] E.J. Neuhold and T. Olnhoff - "The Vienna Definition Method (VDM) and its use for the specification of a relational data base system" - in 'Information Processing 80' - S. Lavington (ed.) - North-Holland (1980).

[Pag] F.G. Pagan - "Formal specification of programming languages" - Prentice-Hall (1981).

[Pao] P. Paolini - "Verification of views and application programs" - Proc. Workshop on Formal Bases for Databases, Toulouse (1979).

[Par] D.L. Parnas - "The use of precise specifications in the development of software" - in 'Information Processing 77' - B. Gilchrist (ed.) - North-Holland (1977) 861-867.

[RB] D. Ridjanovic and M.L. Brodie - "Defining database dynamics with attribute grammars" - Information Processing Letters, vol.14, n. 3 (1982) 132-138.

[Rou] N. Roussopoulos - "CSDL: a conceptual schema definition language for the design of data base applications" - IEEE Trans. on Software Enginnering, SE-5,5 (1979) 481-496.

[Row] L.A. Rowe - "Data abstraction from a programming language viewpoint" - in [BZ] 29-35.

[Sch] D. Schwabe - "Formal techniques for specification and verification of protocols" - Ph.D. dissertation, University of California, Los Angeles (1981).

[Sco] D. Scott - "Outline of a mathematical theory of computation" - Proc. Fourth Annual Princeton Conference on Information Sciences and Systems (1970) 169-176.

[Sh] M. Shaw - "The impact of abstraction concerns on modern programming languages" - Proc. of the IEEE, 68, 9 (1980) 1119-1130.

[Si] E.H. Sibley - "Database management systems: past and present" - in [BZ], p. 192.

[SFNC] U. Shiel, A.L. Furtado, E.J. Neuhold and M.A. Casanova - "Towards multi-level and modular conceptual schema specifications" - to appear in Information Systems.

[SMF] C.S. dos Santos, T.S.E. Maibaum and A.L. Furtado - "Conceptual modelling of data base operations" - International Journal of Computer and Information Sciences, 10, 5 (1981) 299-314.

[SS] J.M. Smith and D.C.P. Smith - "Database abstractions: aggregation and generalization" - ACM TODS 2, 2 (1977) 105-133.

[Su] B. Sundgren - "A conceptual foundation for the infological approach to data bases" - in 'Data base management' - J.W. Klimbie and K.L. Koffeman (eds.) - North-Holland (1974) 61-94.

[Te] R.D. Tennent - "The denotational semantics of programming languages" - Comm. of the ACM - 19, 8 (1976) 437-453.

[TF] T.J. Teorey and J.P. Fry - "Design of database structures" - Prentice-Hall (1982).

[TK] D.C. Tsichritzis and A.Klug (eds.) - "The ANSI/X3/SPARC DBMS framework - report of the study group" - AFIPS Press (1977).

[TL] D.C. Tsichritzis and F.L. Lochovsky - "Data models" - Prentice-Hall (1982).

[VCF] P.A.S. Veloso, J.M.V. de Castilho and A. L. Furtado - "Systematic derivation of complementary specifications" - Proc. Seventh International Conference on Very Large Data Bases (1981) 409-421.

[VF] P.A.S. Veloso and A.L. Furtado - "Multi-level specifications based on traces" - Proc. International Computer Symposium on Application Systems Development, Nurnberg (1983, to appear).

[Wi] A. van Wijngaarden et al (eds.) - "Revised report on the algorithmic language ALGOL 68" - Acta Informatica, 5 (1975) 1-236.

[WPP] D.H.D. Warren, L.M. Pereira and F. Pereira - "PROLOG - the language and its implementation compared with LISP" - Proc. Symposium on AI and Programming Languages - SIGART Newsletters, 64, Aug. (1977) 109-115.

[Zi] S.N. Zilles - "Types, algebras and modelling" - in BZ 207-209.