FORMAL DATA BASE SPECIFICATION - AN ECLECTIC PERSPECTIVE

Marco A. Casanova*, Paulo A.S. Veloso**, Antonio L. Furtado**

* Centro Científico de Brasília
IBM do Brasil
Caixa Postal 853
70.000, Brasilia, DF
Brasil

** Departamento de Informática
Pontifícia Universidade Católica do RJ
Rua Marques de S. Vicente, 209
22.453, Rio de Janeiro, RJ
Brasil

ABSTRACT

Logical, algebraic, programming language, grammatical and denotational formalisms are investigated with respect to their applicability to formal data base specification. On applying each formalism for the purpose that originally motivated its proposal, it is shown that they all have a fundamental and well integrated role to play in different parts of the specification process. An example is included to illustrate the methodological aspects.

1. INTRODUCTION

Although data base theory has been largely influenced by concepts derived from first-order logic, either in their pure form or adapted to the particular needs of data base research, there have been many attempts to use algebra, high-level programming language constructs, grammars and denotational semantics to capture data base concepts. The purpose of this paper is to investigate the applicability of these different kinds of formalisms to the process of specifying data base applications subjected to integrity constraints.

The major contribution of the paper lies in selecting the appropriate variation of each formalism for each level of specification, in the style of organizing the formalisms together into a coherent conceptual design framework and in the formal notion of refinement binding the different levels. Thus, contrarily to most published literature, we neither limit ourselves to just one formalism at just one level nor force the use of the same formalism at different levels, which often creates distortions. Finally, although the paper is not intended to be a survey of the area, it may serve as a guide to different approaches to data base theory.

2. THE CONCEPTUAL DESIGN FRAMEWORK

We divide the design process into three levels of specification, which can be summarized as follows. The first level, the information level, characterizes the data base by its information contents independently of how the information will be used or represented. It gives a high-level description of the set of consistent data base states and the set of state transitions and typically involves a language to talk about the data base and a set of static constraints indicating which states are considered consistent, and a set of transition constraints indicating in turn which transitions are acceptable. In this paper, we will adopt an extension of first-order languages, as described in Section 3.

At the second level, the functions level, we add to the characterization of a data base a repertoire of functions, establishing how we intend to use the information. These functions indicate how the data base will be queried or updated and depend on the applications the designer anticipates for the data base. We will use in this paper an algebraic formalism related to abstract data types, which is described in Section 4.

The third and final level, the representation level, specifies the data base with the help of a data model. A representation of the data base in terms of the data structures supported by the data model must be found and the functions defined at the second level must be mapped into procedures using a Data Manipulation Language (DML) associated with the model. The third level therefore brings us close to the implementation of the data base application on top of a Data Base Management System (DBMS). A programming language, described in Section 5, will be used to specify the data base at the third level. The syntax of the language is given by a grammatical formalism, W-grammars, and its semantics is described using a denotational formalism.

Each level of specification must be a refinement of

the previous one, in the sense that the second-level update functions must preserve the first-level static and transition constraints, and the third-level procedures defining second-level functions must satisfy the second level equations. This is further discussed in Sections 4.3 and 5.3.

The reader is referred to the full version of the paper [VCF] for a more thorough discussion.

3. THE INFORMATION LEVEL - THE USE OF LOGICAL FORMALISMS

3.1 Logical Formalisms

In this section we briefly indicate how a data base can be specified, at the information level, using a logical formalism. We assume familiarity with first order logic at the level, say, of [En], so that the presentation of the formalism will be very terse.

There have been attempts to either use subsets of first-order logic or use extensions of the formalism depending on the aspects of data base design in question. We illustrate this phenomenon in this paper by defining an extension of first-order languages that helps capturing transition constraints. The extension is perhaps the simplest one and depends on the introduction of two modal operators. Other sets of modal operators can be adopted to enhance the expressive power of the language. A different approach could also be taken by selecting a many-sorted first order language with a special sort interpreted as time (see [CF,BADW] for extensive discussions).

Given a (many-sorted) first-order language L, its temporal extension, LT, is defined as follows. The symbols of LT are those of L, plus one modal operator , the possibility operator denoted by $\Diamond$. The modal operator $\Box$ of necessity is the dual of $\Diamond$ in the sense that it can be introduced by definition as $\Box P \equiv \neg \Diamond \neg P$. The terms of LT are those of L and the set of wffs of L is defined using the familiar formation rules, plus one new rule:

If P is a wff of L or LT, then $\Diamond P$ is also a wff of LT

The semantics of LT is defined as follows. A universe U for LT is a pair (S,R), where S is a set of structures of L, all with the same domain D (this restriction can be relaxed, but it simplifies the treatment of quantifiers), and R is a binary relation over S, called the accessibility relation. Given a wff P of LT, a structure A in S and a valuation v over the common domain D, we define the notion that A satisfies P with v in U (denoted $\models^A_U P[v]$) using rules identical to those of first-order languages, plus one additional rule:

$\models^A_U (\Diamond P)[v]$ iff there is B in S such that R(A,B) and $\models^B_U P[v]$

The notions of model, logical implication and theory are as for first-order languages.

Thus, to account for transition constraints, a data base is specified at the information level by defining a theory $T_1=(L_1,A_1)$, where L_1 is a temporal extension of a (many-sorted) first-order language L and A_1 is a set of axioms. The non-logical symbols of L_1 describe the data base data structures and all ordinary symbols such as "less than", used to express facts about the data base. Symbols representing data base structures are called db-predicate symbols. The axioms in A_1 define static constraints, if they do not involve modalities, or transition constraints, otherwise. The semantics of the data base is fixed by selecting a universe U=(S,R) for L_1. The structures in S play the role of data base states and the relation R over S is interpreted as indicating that, if (A,B) is in R, then B is a future state with respect to A. A structure A in S corresponds to a consistent state iff it is a model of A_1.

We note that the semantics of a data base, as explained above, is only loosely fixed by the theory T_1, especially the relation R. This situation is modified when the functions level (i.e., algebraic) specification of the data base is fixed (section 4).

3.2 An Example

We are now in a position to present our example data base and formalize it at the information level.

The example data base is defined by a theory $T_1=(L_1,A_1)$, where L_1 is a many-sorted temporal language with two sorts, course and student, and two predicate symbols, *offered*, of sort course , and *takes*, of sort <student, course>. The intended interpretation of *offered*(c) is that course c is offered, and of *takes*(s,c) is that student s takes course c. The set A_1 of axioms consists of two formulas:

(1) $\neg\exists s \exists c(takes(s,c) \wedge \neg offered(c))$
(2) $\neg\exists s \exists c(\Diamond(takes(s,c) \wedge \Diamond(\neg\exists c' takes(s,c'))))$

The first formula formalizes the static constraint: "a student cannot take a course that is not being offered". The second formula formalizes the transition constraint: "the number of courses taken by a student cannot drop to zero" (i.e., he cannot be taking a course in (some) current state and no course in a future state).

4. THE ALGEBRAIC LEVEL - THE USE OF ALGEBRAIC FORMALISMS

4.1 Algebraic Formalisms

Recall that the goal of a second level specification is to define a set of query and update functions that preserve the static and transition constraints listed at the information level specification, provided that only such functions be used (the encapsulation strategy). This can be achieved by giving the data base application an algebraic specification [VF,DMW].

An algebraic specification is a first-order theory T=(L,A), where L is a many-sorted first-order language and A is a set of axioms obeying the following restrictions.

The set of sorts of L must include a Boolean sort and a designated sort state (also called sort-of-interest). The remaining sorts are called parameter sorts. The only predicate symbols of L are two equality symbols of sorts <Boolean,Boolean> and <state,state>. For simplicity, and since no ambiguity arises, both are denoted by '='. The parameter sorts

of L are endowed with their own function symbols (not involving the sort state) which have the effect of generating a set of ground terms called parameter names. Besides, each parameter sort s must have a function symbol of sort <s,s,Boolean>, also denoted by '=', which checks equality among objects of that sort.

The Boolean sort will be equipped with two constants, True and False, and with five function symbols standing for the usual connectives, $\neg$, $\vee$, $\wedge$, $\Rightarrow$, $\equiv$ written in infix notation.

The language L may have other function symbols as long as state occurs as one of the domain sorts. To simplify the notation, we assume that state is always the last one in the list of domain sorts. Thus, if f is an n-ary function symbol in this group, it must have a sort of the form $<s_1,\ldots,\text{state},s_{n+1}>$ (recall that s_{n+1} is the target sort). If s_{n+1} is the sort state then f is an update function (intuitively, it maps states into states according to some arguments); otherwise, f is a query function (it interrogates the current state, according to some arguments and returns a value). Let f be an n-ary query function. Whenever terms of sorts other than state are irrelevant, we will write f(s) instead of $f(t_1,\ldots,s)$.

A term of the form $q(t_1,\ldots,t_n)$ where q is a query function and $t_1,\ldots,t_n$ contain no occurrences of update functions is called a simple observation. We will construct the language L_2 to be sufficiently rich with queries so that states can be identified by means of simple observations. More precisely, if s and s' are state variables such that for all simple observations f we have f(s)=f(s'), then s=s'. This observability condition is often fulfilled by data base applications due to their purpose.

The type of axioms allowed in algebraic specifications will be conditional equations, which are wffs of the form $P \Rightarrow t=t'$ where P is a wff and t and t' are terms of the same sort s. If s is state then we call the axiom an U-equation, otherwise we call the axiom a Q-equation. Often term t' is "simpler" than t and we can view an axiom as a conditional term-rewriting rule.

An algebraic specification, being a theory, defines a set of structures, the models of the theory. (In the context of algebraic specifications, structures are called (many-sorted) algebras). As usual, we further restrict this set to be the set of all finitely generated algebras (i.e., those in which every element is the value of a variable-free term) which are models of the axioms. Thus, we can employ the principle of structural induction (on terms) as a proof rule.

We call an algebraic specification T=(L,A) sufficiently complete iff for every ground term of the form $q(t_1,\ldots,t_n)$, where q is a query function (with target sort s, say), there exists a parameter name p (of sort s) such that $\vdash_A q(t_1,\ldots,t_n) = p$. Intuitively, a sufficiently complete algebraic specification is one enabling the evaluation of all queries.

4.2 Obtaining a Functions Level Specification - An Example

We now outline the methodology we employ to obtain an algebraic specification $T_2=(L_2,A_2)$ of a data base application at the functions level.

Consider again the data base application described at the information level by the theory $T_1=(L_1,A_1)$ of Section 3.2. For simplicity, we take the parater sorts of L_2 as the sorts of L_1. Moreover, we correlate the db-predicate symbols of L_1 describing data base structures with query function symbols. So, L_2 will contain two query function symbols, offered and takes, of sorts <course,state,Boolean> and <student,course,state,Boolean>, respectively. The intended interpretation of offered(c,σ), for example, is that it is true iff c is a course offered in state σ.

The update function symbols (with their intended interpretation) are: initiate of sort <state>, with initiate understood as an operation that initializes the data base; offer of sort <course,state,state>, where offer(c,σ)=δ indicates that c is added as a new course to state σ, creating state δ; cancel of sort <course,state,state>, where cancel(c,σ)=δ means the inverse operation; enroll of sort <student,course,state,state>, where enroll(s,c,σ)=δ creates a new state δ by enrolling student s to course c on state σ; transfer of sort <student, course,course,state,state>, with transfer(s,c,c', σ)=δ understood as creating state δ from state σ by transferring student s from course c to course c'.

Our task now is to write a set of conditional equations from which we can obtain the correct result of every query and, at the same time, guarantee that consistency is always preserved. In other words, for every query function q, for all parameters p and for all ground terms t of sort state, we should be able to derive from the axioms the equality q(p,t)=b where the Boolean value b is the correct answer according to the given description. Now, the set T of ground terms of sort state is the smallest set of terms containing initiate and closed under symbolic application of the other update functions. Thus, we shall strive for Q-equations of the form (perhaps with some condition):

$$q(p,u(p',\sigma)) = \text{"simpler expression"}$$

for all query functions q, update functions u and parameter lists p and p', σ being a variable of sort state.

In order to obtain such equations we employ structured descriptions giving, for each update function, its intended effects, preconditions for state change, possible side-effects, and simple observations that are not affected. In fact, we obtain equations that are guaranteed, by construction, to be correct with respect to the description. Then, we verify sufficient completeness.

As an example of the method, let us consider the update function cancel, whose structured description is:

δ = cancel(c,σ)

/* course c is cancelled at state δ, providing that no student is taking it at state σ */
intended effects: offered(c,σ) = False
pre-conditions: $\forall$s(takes(s,c,σ) = False)
side-effects: none
not-affected: all other queries, including offered(c',.) with c'$\neq$c

We shall examine in detail the case of the query offered. In other words, we want (conditional) equations enabling us to derive the correct results of queries of the form:

offered(c',cancel(c,σ))

We shall d vide our task into two cases depending on the comparison of c' with c.

For the first case (c$\neq$c') the not-affected part of the structured description tells us that the value of offered(c',.) is not affected by the update, i.e.

offered(c',cancel(c,σ)) = offered(c',σ)

We can put this into the form of a conditional equation

c'$\neq$c $\Rightarrow$ offered(c',cancel(c,σ)) = offered(c',σ)

Notice that the antecedent of the conditional equation does not involve terms of sort state, only parameters. Also, the righthand side of the consequent is "simpler" than the lefthand side.

Now let us examine the case c'=c. According to the structured description, the value of offered(c,cancel(c,σ)) will depend on the pre-condition. If the pre-condition holds then we have the intended effect False. Otherwise the value remains unchanged. Thus, we have:

offered(c,cancel(c,σ)) =
False if $\forall$s(takes(s,c,σ) = False)
offered(c,σ) if $\exists$s(takes(s,c,σ) = True)

Now, in view of the static constraint, we have:

$\exists$s(takes(s,c,σ) = True) $\Rightarrow$ offered(c,σ) = True

So, we can write

offered(c,cancel(c,σ)) = True $\equiv$ $\exists$s(takes(s,c,σ) = True)

which can be rewritten as two conditional equations:

$\exists$s(takes(s,c,σ) = True) $\Rightarrow$ offered(c,cancel(c,σ))=True

$\neg\exists$s(takes(s,c,σ)=True) $\Rightarrow$ offered(c,cancel(c,σ))=False

Three remarks are in order. First, in obtaining this equation we used the static constraint (assumed to hold; we shall later have to verify that it does hold). Second, the antecedents of the above conditional equations do not involve quantification over states, only over parameters. Third, we may regard these equations as reducing the problem of determining offered(c,cancel(c,σ)) to that of determining whether there exists a student s such that takes(s,c,σ)=True, which may be viewed as a problem somewhat simpler than the original one. However we must be careful, for some other equation might reduce the problem of determining takes(s,c,σ) to that of determining offered(c,σ), thereby creating a circularity. This is the reason why we later verify termination.

By applying the general methodology outlined above we obtain the following set of Q-equations for our example:

1. offered(c,initiate) = False
2. takes(s,c,initiate) = False
3. offered(c,offer(c,σ)) = True
4. c$\neq$c' $\Rightarrow$ offered(c,offer(c',σ)) = offered(c,σ)
5. takes(s,c,offer(c',σ)) = takes(s,c,σ)
6. offered(c,cancel(c,σ)) = True $\equiv$ $\exists$s(takes(s,c,σ) = True)
7. c$\neq$c' $\Rightarrow$ offered(c,cancel(c',σ)) = offered(c,σ)
8. takes(s,c,cancel(c',σ)) = takes(s,c,σ)
9. offered(c,enroll(s,c',σ)) = offered(c,σ)
10. takes(s,c,enroll(s,c,σ)) = offered(s,σ)
11. s$\neq$s' $\vee$ c$\neq$c' $\Rightarrow$ takes(s,c,enroll(s',c',σ)) = takes(s,c,σ)
12. offered(c,transfer(s,c',c",σ)) = offered(c,σ)
13. takes(s,c',transfer(s,c,c',σ)) = (offered(c',σ) $\wedge$ takes(s,c,σ)) $\vee$ takes(s,c',σ)
14. takes(s,c,transfer(s,c,c',σ)) = ($\neg$offered(c',σ) $\vee$ takes(s,c',σ)) $\wedge$ takes(s,c,σ)
15. s$\neq$s' $\vee$ (c$\neq$c' $\wedge$ c$\neq$c") $\Rightarrow$ takes(s,c,transfer(s',c',c",σ)) = takes(s,c,σ)

4.3 First to Second Level Refinements

The information and functions level specification of a data base application are bound by a notion of refinement we describe in this section. Let T_1=(L_1, A_1) and T_2=(L_2,A_2) be the information and functions level specification of the same data base application. Intuitively, we say that T_2 refines T_1 iff the axioms in A_2 are sufficient to guarantee that the updates preserve consistency with respect to the static and transition constraints in A_1. Although this condition is on the surface simple, it creates some technical difficulties to be formalized, mainly because the two languages, L_1 and L_2, are of different types. In particular, wffs of L_1 may contain modalities, which are not part of L_2.

For simplicity, we assume that every sort of L_1 is a parameter sort of L_2 and every variable of L_1 is also a variable of L_2.

The notion of refinement is formally defined by specifying an interpretation I mapping the non-logical symbols of L_1 into terms of L_2 with the following characteristics:

(1) for each n-ary db-predicate symbol p of sort $\langle s_1,\ldots,s_n\rangle$ of L_1, I(p) must be a term of L_2 of sort Boolean and free variables $x_1,\ldots,x_n,y$ of sorts $s_1,\ldots,s_n$,state;

(2) for each other n-ary predicate symbol p of sort $\langle s_1,\ldots,s_n\rangle$ of L_1, I(p) must be a wff of L_2 with free variables $x_1,\ldots,x_n$ of sorts $s_1,\ldots,s_n$;

(3) for each function symbol f of sort $\langle s_1,\ldots,s_n, s_{n+1}\rangle$ of L_1, I(f) must be a term of L_2 of sort s_{n+1} and free variables $x_1,\ldots,x_n$ of sorts $s_1,\ldots,s_n$.

In our running example, we might define an interpretation I that assigns to the db-predicate symbol *offered* the term offered(c,σ) and to *takes* the term takes(s,c,σ).

Thus, the notion of interpretation defined above follows the general idea of first-order interpretations.

Given an interpretation I, we can extend I to map wffs of L_1 into wffs of L_2. However, in order to do so, we must extend L_2 by adding a predicate symbol F of sort <state,state>, which will stand for the reachability relation R of the semantics of L_1. The extension of I is defined in then full paper and an example appears in the next section.

Thus, at this point we know how to map wffs of L_1 into wffs of L_2. Therefore, we can check if indeed the equations of T_2 are enough to guarantee that all updates of T_2 preserve consistency. More precisely, we say that T_2 is a correct refinement of T_1 iff for any axiom P of T_1, I(P) is a theorem of T_2.

As for first-order languages, our notion of interpretation can also be used to induce a mapping M ffom structures of L_2 into universes of L_1, which permits us to give an alternative (semantical) characterization of correct refinement (see the full paper).

4.4. Proof of Correctness of Refinement - An Example

Let $T_2=(L_2,A_2)$ be the algebraic specification of the data base application obtained in Section 4.2. We must guarantee that T_2 has the following properties:

- it is sufficiently complete and correct with respect to the structured description;
- it is a correct refinement of the first-level specification given in Section 3.2.

By construction our equations are already correct with respect to the structured description. We proceed by proving:

(a) sufficient completeness
(b) static eonsistency, i.e. every reachable state is valid
(c) every valid state is reachable
(d) transition consistency

Parts (b) and (d) are equivalent to saying that the refinement is correct. Part (c) enables us to simplify the verification of part (d), in addition to being an interesting property by itself. Notice, however, that by contrast not all valid transitions will be realized by our repertoire of update functions.

We outline below how these properties can be proven:

(a) Sufficient Completeness

We can view our set of Q-equations as a system of mutually recursive equations defining the query functions. From this viewpoint, sufficient completeness amounts to termination of this system of recursive definitions. There are several criteria for checking termination of such term rewriting systems. However, the basic idea is checking the absence of circularity in these definitions. This basic idea will do for cases simple as our example.

(b) Every reachable state is valid

Consider the set V of all valid states, i.e., the set defined by

$$\forall c \forall s(\underline{takes}(s,c,\sigma)=\underline{True} \Rightarrow \underline{offered}(c,\sigma)=\underline{True})$$

The set G of reachable states is the least set of states containing initiate and closed under all the other update functions. So, in order to show that the static constraint is satisfied at the functions level, i.e., $G \subset V$, it suffices to show that V contains initiate and is closed under all other update functions.

(c) Every valid state is reachable

Consider again the sets V of valid states and G of reachable states. We now want to show the inclusion $V \subset G$. For this purpose we can proceed by induction on the number of courses offered and the number of enrollments of students in courses.

(d) Transition consistency

The transition constraint of our example (see Section 3.2) is logically equivalent to

$$\forall s \forall c(\Box(\underline{takes}(s,c) \Rightarrow \Box(\exists c' \underline{takes}(s,c'))))$$

which can be rewritten, by applying the notion of refinement as

$$\forall \sigma_0(\forall s \forall c \forall \sigma(F(\sigma_0,\sigma) \Rightarrow (\underline{takes}(s,c,\sigma) = \underline{True} = \forall \delta(F(\sigma,\delta) \Rightarrow \exists c' \underline{takes}(s,c',\delta)) = \underline{True})))$$

where F corresponds to the accessibility relation.

We shall first check

$$\forall s \forall c \forall \sigma(\underline{takes}(s,c,\sigma)=\underline{True} \Rightarrow \exists c'(\underline{takes}(s,c',u(\sigma))=\underline{True})$$

for each update function u other than initiate.

We illustrate this checking with the case of cancel. For this purpose notice that, by equation 8

$$\underline{takes}(s,c',\underline{cancel}(c'',\sigma)) = \underline{takes}(s,c',\sigma)$$

So, if takes(s,c,σ) = True then there is c' = c such that takes(s,c',cancel(c",σ)) = True.

The case of offer is entirely similar. For the update functions enroll and transfer the checking can be performed by breaking into cases depending on the comparison of the values of the parameters.

Thus we have that every single-update transition obeys the transition constraint. It follows readily by induction that every transition (effected by a sequence of updates) also obeys the transition constraint.

5. THE REPRESENTATION LEVEL - THE USE OF A PROGRAMMING LANGUAGE FORMALISM

5.1 Programming Language Formalism

To qualify as a specification formalism, a programming language must be simple and theoretically sound. We shall use regular programs over relations (RPR)

(see [CB]), which is associated with the relational model and is based on the concept of regular programs of [Ha].

We note that, by specifying a language associated with a data model, we are in a sense providing a formal specification of the data model itself.

5.1.1 Syntax - The Use of a Grammatical Formalism

Briefly, the syntax of a data base schema is defined as follows. Let L be a many-sorted first-order language with a set of distinguished constants, called scalar program variables. If P is a wff of L with free variables $x_1,\ldots,x_m$, then we call an expression of the form $\{(x_1,\ldots,x_m)/P\}$ a relational term of sort $<s_1,\ldots,s_m>$, if s_i is the sort of x_i.

A data base schema has the following format:

schema SCL ; OPL end-schema

SCL is a list of statements of the form $R[A_1,\ldots,A_n]$ where R is a predicate symbol of L and $A_1,\ldots,A_n$ are unary predicate symbols of L such that, if $<s_1,\ldots,s_n>$ is the sort of R, then A_i has sort $<s_i>$, for each $i=1,\ldots,n$. Each predicate symbol R in SCL is called a relation name or relational program variable. OPL is a list of operation declarations of the form "proc $I(Y_1,\ldots,Y_n)$ = S" where I is an operation identifier, Y_i is either a scalar or a relational program variable, and S is a statement, called the operation body.

The set of statements (based on L), is defined inductively as follows:

(1) For any scalar program variable x of L and any variable-free term t of L of the same sort as x, the expression x := t is an assignment statement;
(2) For any relational program variable R of L and any relational term F of the same sort as R, the expression R := F is a relational assignment statement;
(3) For any closed wff P of L, P? is a test statement;
(4) For any statements p and q, the expressions $(p \cup q)$, (p ; q) and p* are statements called the union of p and q, the composition of p and q and the iteration of p, respectively.

We may also introduce some familiar constructs by definition such as if-then, if-then-else, while, insert and delete. Statements constructed using these statements and assignments are called deterministic.

The formal definition of the syntax of data base schemas is given in the full paper, using W-grammars (see also [FVC]). W-grammars (as also other comparable formalisms, such as attribute grammars and affix grammars) go beyond BNF in that they can express context-sensitive restrictions (e.g., that all relational program variables in the OPL part of a schema have been declared in the SCL part), and can be used to build compiler generators. A correspondence between W-grammars and logic has been established in [He].

5.1.2 Semantics - The Use of a Denotational Formalism Formalism

Let L be the underlying many-sorted first-order language. For a given structure A of L and a given non-logical symbol s of L, let A(s) denote the value of s in A. Likewise, let A(t) be the value of a variable-free term t of L in A and let A(F) be the relation denoted by F, if F is a relational term.

A universe U for L is a set of structures of L satisfying three conditions:

(i) any two structures in U differ only on the values of the scalar or relational program variables;
(ii) for any A in U, any scalar program variable x and any element e of the domain of the sort of x, there is B in U such that A and B differ only on the value of x, which is e in B;
(iii) for any A in U, any relational program variable R of sort $<s_1,\ldots,s_n>$ and any n-ary relation $r \subset D_{s_1}\times\ldots\times D_{s_n}$, where D_{s_i} is the domain of sort s_i, there is B in U such that A and B differ only on the value of R, which is r in B.

For a fixed universe U of L, the meaning of statements is given by a function m assigning to each statement in RPR a binary relation in U as follows:

(1) $m(x:=t) = \{(A,B) / B \text{ is equal to A, except that } B(x) = A(t)\}$
(2) $m(R:=\{(x_1,\ldots,x_n) / P\}) = \{(A,B) / B \text{ is equal to A, except that } B(R) = A(F)\}$
(3) $m(P?) = \{(A,A) / P \text{ is true in A}\}$
(4) $m(p \cup q) = m(p) \cup m(q)$ (union of both relations)
(5) $m(p ; q) = m(p) \circ m(q)$ (composition)
(6) $m(p^*) = (m(p))^*$ (closure of m(p))

The meaning of procedure declarations is given by a function k assigning to each procedure declaration d of the form proc $I(Y_1,\ldots,Y_m)$= S a function from $D_{s_1}\times\ldots\times D_{s_m}$ into the set of all binary relations over the universe, where D_{s_i} is the domain of of sort s_i and Y_i is of sort s_i. The function k is defined as follows:

(7) $k(d) = f$ iff for any $(c_1,\ldots,c_m)$ in $D_{s_1}\times\ldots\times D_{s_m}$ $f(c_1,\ldots,c_m)$ is the set of all pairs (A,B) in $U\times U$ such that $(A[c_1/Y_1,\ldots,c_m/Y_m],B)$ is in m(S)

If the procedure bodies are deterministic programs, then the range of k is the set of all functions from U into U.

The formal definition of the functions m and k is given in the full paper using the denotational approach [BJ].

5.2 Obtaining a Representation Level Specification - An Example

Obtaining the third level specification means to express in the programming language introduced in the previous section both the kinds of predicates to be used, under the guise of relations, and the

query and update functions that will act upon them. Query functions are trivially introduced by noting that the language allows logical-valued expressions of the form R(t) that yield True if t is in R, and False otherwise.

In order to obtain in a constructive manner procedures that implement the desired update functions, we first correlate the four parts of our structured (semi-formal) description of update functions with the semantics of the statements of the programming language.

From the semantic definitions, one readily sees that, in the simpler cases, an update function f will follow the pattern:

proc f($\bar{x}$) = (pre-conditions?;effects;side-effects)
∪ ¬pre-conditions?

which can also be written using the if-then construct.

More complex updates may require (possibly nested) tests and iterations. The latter are useful, in particular, to check a universally quantified pre-condition. Explicitly quantified pre-conditions and the general form of assignments lead to a more "set-oriented" style of programming, whereas the use of iteration and insert/delete statements favor a "tuple-oriented style.

The complete programming language specification for the example is given below:

schema

```
OFFERED(Students);
TAKES(Students, Courses);
proc initiate() =
     (TAKES := ∅ ; OFFERED := ∅)
proc offer(c) =
     insert OFFERED(c)
proc cancel(c) =
     if ¬∃s TAKES(s,c)
        then delete OFFERED(c)
proc enroll(s,c) =
     if OFFERED(c)
        then insert TAKES(s,c)
proc transfer(s,c,c') =
     if TAKES(s,c) ∧ ¬TAKES(s,c') ∧ OFFERED(c')
        then (delete TAKES(s,c);
              insert TAKES(s,c'))
```

end-schema

5.3 Second to Third Level Refinements

Let $T_2 = (L_2, A_2)$ and T_3 be the functions and representation level specifications of the same data base application. Then, the operations defined by procedures in T_3 must satisfy all equations in A_2. Again, we must face the fact that T_2 and T_3 use different formalisms, so we do not have a notion of interpretation readily available.

Recall that T_3 uses a programming language, which is in turn based on a first-order language, say, L_3. For simplicity, we assume that every parameter sort of L_3 and every variable of sort s of L_2 is also a variable of L_3.

The notion of refinement is again formally defined by specifying a mapping K from the non-logical symbols of L_2 into non-logical symbols of L_3, wffs of L_3 and procedure declarations of T_3. The mapping K must satisfy the following requirements:

(1) for each n-ary update function symbol u of L_2 of sort $\langle s_1,\ldots,s_{n-1},\text{state},\text{state}\rangle$ K(u) is a procedure declaration proc $U(Y_1,\ldots,Y_{n-1}) = S$ in T_3 such that Y_i is of sort s_i, for i=1,..., n-1.

(2) for each n-ary query function symbol q of L_2 of sort $\langle s_1,\ldots,s_{n-1},\text{state},\text{Boolean}\rangle$, K(q) is a wff of L_3 with free variables $x_1,\ldots,x_{n-1}$ of sorts $s_1,\ldots,s_{n-1}$

(3) for each n-ary function symbol f of L_2 of sort $\langle s_1,\ldots,s_n,\text{Boolean}\rangle$, except those above and those representing logical connectives and Boolean constants, K(f) is a wff of L_3 with free variables $x_1,\ldots,x_n$ of sorts $s_1,\ldots,s_n$.

(4) for each n-ary function symbol f of L_2 of sort $\langle s_1,\ldots,s_n,s_{n+1}\rangle$ with s_{n+1} not equal to Boolean or state, K(f) = f.

note: the requirement in (5) could be generalized to K(f) being a wff of L_3 with free variables $x_1,\ldots,x_n,y$ of sorts $s_1,\ldots,s_{n+1}$, if we could force the wff K(f) to define a function as for first-order interpretations.

We now pause for a comment from our formalism department. If the reader remembers section 4.3, the next natural step would be to extend K to map wffs of L_2 into wffs of L_3. However, L_3 is not powerful enough to permit us to carry on such extension. In order to do so, we would need a full programming logic, such as Dynamic Logic (a separate paper will explore this possibility). To circumvent this difficulty, we adopt a semantic definition of correct refinement.

Thus, using an interpretation K, we define a mapping N from universes of L_3 into finitely generated structures of L_2 (see full paper).

Now, using N, we say that T_3 is a correct refinement of T_2 iff for every universe of L_3, N(U) is a model of T_2.

5.4 Proof of Correctness of the Refinement - An Example

On analysing the constructive strategy (section 5.2) we observe that the semi-formal considerations that resulted in the algebraic equations of the second level were used but not the equations themselves. Similarly, our understanding of the syntax and semantics of the programming language helped us, but the formal definition of these notions were not directly used.

The formal definitions of the syntax and semantics of the programming language are necessary when we want to prove that the third-level specification is correct. We illustrate this process by taking the specification of Section 5.2 as example.

In order to verify that the specification in Section 5.2 is syntactically correct, we have to guarantee that it can be generated by the corresponding W-grammar, which creates no difficulty (see full paper).

We now outline how we can verify that the represen tation level specification T_3 (see Section 5.2) is a correct refinement of the functions level specification $T_2 = (L_2, A_2)$ (see Section 4.2) under the interpretation K defined below:

K(offered) = OFFERED(c)
K(takes) = TAKES(s,c)
K(u) = U , where u is an update function and U is the homonym procedure

Let L_3 be the underlying language of T_3.

Intuitively, given a universe U for T_3, the interpretation K induces a finitely generated structure A for L_2. At this point, it suffices to clarify that each element p of the domain of the sort state of A will be in fact a structure in U. From now on, we will refer to such elements simply as states and use p,q,r,... with subscripts if necessary to denote them (the reader must bear in mind that states are structures of L_3).

It is also important to stress that the domain of sort state is finitely generated by construction. That is, each element p of this domain is the value of a term of L_2, which is schematically of the form:

$$u_n(u_{n-1}(\ldots u_1(u_o)\ldots))$$

where u_0 is the update function symbol initiate of L_2 and u_i with i=1,...,n are also update function symbols of L_2.

Intuitively, since the data base application is encapsulated by the query and update functions, the current data base state can be represented by the sequence-composition (trace) of the operations used thus far.

To prove that T_3 is a correct refinement of T_2 amounts to proving that each of the conditional equations in A_2 is (universally) valid in A. Now, since A is finitely generated and in view of the previous discussion, we can in fact do an induction on the lenght of the term $u_n(u_{n-1}(\ldots(\underline{initiate})\ldots))$ corresponding to each element p of the domain of sort state of A. That is, for each P in A_2, we will prove by induction on n that P is valid in A when the state variable σ receives as value some state p and p is in turn the value of a term $u_n(u_{n-1}(\ldots(\underline{initiate})\ldots))$ of L_2.

The basis is trivial. So assume that each P in A_2 is valid in A when the state variable receives as value some state r and r is the value of a term $v_{n-1}(v_{n-2}(\ldots(\underline{initiate})\ldots))$ of L_2. We will shown that the result holds when we consider terms of length n.

Now, let q be an element of S and assume that q is the value of a term $u_n(u_{n-1}(\ldots(\underline{initiate})\ldots))$ of L_2.

As an example of the induction hypothesis case analysis, consider equation 6, namely,

(1) $\underline{offered}(c,\underline{cancel}(c,\sigma)) = \underline{True} \equiv \exists s(\underline{takes}(s,c,\sigma) = \underline{True})$

In view of the construction of A (which the reader will find in the full paper), equation 6 is universally valid in A when σ is valued as q iff the following condition holds (from now on, C will denote the domain of sort course of A, T will denote the domain of sort student of A and S will denote the domain of sort state of A):

(2) for each $b \in C$, $b \in q(\text{OFFERED})$ iff $(t,b) \in p(\text{TAKES})$, for some $t \in T$ where $(p,q) \in k[\![\text{cancel}(c)]\!](b)$

note: q(OFFERED) denotes the value of OFFERED in the structure q, and similarly for p(TAKES)

Now, by definition of the procedure cancel, we have:

(3) $(p,q) \in k[\![\text{cancel}(c)]\!](b)$ iff
(3.1) if $(t,b) \notin p(\text{TAKES})$, for any $t \in T$ then q is equal to p, except that $q(\text{OFFERED}) = p(\text{OFFERED}) - \{b\}$

or

(3.2) if $(t,b) \in p(\text{TAKES})$, for some $t \in T$ then q is equal to p

In view of the form of (3) (consisting of the disjunction of two conditionals) it is natural to divide this verification into two cases:

case 1: $(t,b) \notin p(\text{TAKES})$, for any $t \in T$.

Thus, by (3.1), state q is equal to p except that $q(\text{OFFERED}) = p(\text{OFFERED}) - \{b\}$. Therefore, $b \notin q(\text{OFFERED})$, which suffices to establish (2) in view of the conditions of the case.

case 2: $(t,b) \in p(\text{TAKES})$, for some $t \in T$.

Then, by (3.2), state q is equal to p. Thus, $b \in q(\text{OFFERED})$ iff $b \in p(\text{OFFERED})$. We will now show that $b \in p(\text{OFFERED})$, under the assumption that $(t,b) \in p(\text{TAKES})$, for some $t \in T$.

Recall that q is the value of a term $u_n(u_{n-1}(\ldots(\underline{initiate})\ldots))$. Let r_i be the state denoted by the term $u_i(u_{i-1}(\ldots(\underline{initiate})\ldots))$, for i=1,...,n (hence r_n=q). By the induction hypothesis, each equation P in A_2 is valid in A when σ is valuated as r_i and c is valuated as b, for i=1,...,n-1. (We will use $A \models P[b/c, r_i/\sigma]$ to indicate this condition). Let us proceed in a backward direction to examine the various possibilities for each u_i, for any $b',b'' \in C$ and $t,t' \in S$:

(4.1) if u_i is initiate then, by equation 2,
$A \models (\underline{takes}(s,c,\underline{initiate}) = \underline{False})[t/s,b/c]$

(4.2) if u_i is offer then, by equation 5,
$A \models (\underline{takes}(s,c,\underline{offer}(c',\sigma)) = \underline{takes}(s,c,\sigma))[t/s,b'/c',b/c,r_{i-1}/\sigma]$

(4.3) if u_i is cancel then, by equation 8,
$A \models (\underline{takes}(s,c,\underline{cancel}(c',\sigma)) = \underline{takes}(s,c,\sigma))[t/s,b'/c',b/c,r_{i-1}/\sigma]$

(4.4) if u_i is enroll then, by equation 10,
$A \models (\underline{takes}(s,c,\underline{enroll}(s,c,\sigma)) = \underline{offered}(c,\sigma))[t/s,b/c,r_{i-1}/\sigma]$

and by equation (11), if $t \neq t'$ and $b \neq b'$:

$$A \models (\underline{takes}(s,c,\underline{enroll}(s',c',\sigma)) = \underline{takes}(s,c,\sigma))[t/s,t'/s',b/c,b'/c',r_{i-1}/\sigma]$$

(4.5) if u_i is transfer then, by equations 13,14,15

$$A \models ((\underline{takes}(s,c,\underline{transfer}(s',c',c'',\sigma)) \Rightarrow (\underline{offered}(c,\sigma) \vee \underline{takes}(s,c,\sigma)) = \underline{True})[t/s,t'/s',b/c,b'/c',b''/c'',r_{i-1}/\sigma]$$

The backward induction uses (4.1)-(4.5) repeatedly. In many cases we are simply led to examine a previous state , since the expression says that b is offered after the application of u_i if it was offered at r_{i-1}. However this process cannot reach initiate, where b would not be offered, contrarily to the condition of case 2. We can verify that the only way to fulfill this condition, rewritten as

(5) $A \models (\exists s\ \underline{takes}(s,c,\sigma))[b/c,r_{n-1}/\sigma]$

is either by enrolling some t in b or by transferring some t to b, in any case in a state r_{i-1} where b is offered. Moreover, by equations 9 and 12, b will still be offered after any of the two operations is applied. Hence, we conclude that

(6) there exists $j < n$ such that

$A \models (\exists s(\underline{takes}(s,c,\sigma) = \underline{True})[b/c,r_{n-1}/\sigma]$ iff

$A \models (\exists s(\underline{takes}(s,c,\sigma) = \underline{True})[b/c,r_j/\sigma]$ and

$A \models (\underline{offered}(c,\sigma) = \underline{True})[b/c,r_j/\sigma]$

Let k be the minimum such j. So, from (5) and (6), we have that:

(7) $A \models (\exists s\ \underline{takes}(s,c,\sigma) = \underline{True})[b/c,r_k/\sigma]$ and

$A \models (\underline{offered}(c,\sigma) = \underline{True})[b/c,r_k/\sigma]$

Now, we can proceed in a forward direction to show that indeed

(8) $A \models (\underline{offered}(c,\sigma) = \underline{True})[b/c,r_{n-1}/\sigma]$

since the only way to reach from r_j a state where b is no longer offered is by cancelling b, which fails as long as there is a student taking b (here we are using, among others, equation 6, the very equation that we are about to prove; this is legitimate because we are assuming by hypothesis its validity up to state r_{n-1}).

That is, using the notation of T_3 and the construction of A, and since $p = r_{n-1}$, we have that $b \in p(OFFERED)$, as was to be shown.

Proceeding similarly we can verify that all the equations of the functions level are satisfied by our specification of the representation level.

6. CONCLUSIONS

In spite of marked differences in notation, the five formalisms discussed in this paper have a characteristic in common: they are all related to logic.

The one-to-one correspondence between db-predicate symbols (first level), query functions (second level) and relation names (third level) provided a certain uniformity that facilitated going through the different notations. This coincidence, although not a mandatory design decision, proved to be convenient.

One of the more significant differences across the three levels of specification is the treatment of states. States are implicitly described by their properties at the information level. They are explicit parameters at the functions level. At the representation level they are defined in terms of the value of the entire collection of data base relations; each statement mentions only the relations that it affects. Intermediate states may be considered as an operational (machine-like) aspects of the representation level, resulting from the execution of single statements.

We believe that the discussion and the example substantiate the claim that each formalism does play a relevant role in the formal specification of data bases, especially when used for the objective that originally motivated its proposal.

Selected References

[BADW] A. Bolour, T.L. Anderson, L.J. Dekeyser and N.K.T. Wong, "The Role of Time in Information Processing - A Survey", ACM SIGMOD Record 12,3 (1982), 27-50.

[BJ] D. Bjorner and C.B. Jones, Formal Specification and Software Development, Prentice Hall (1982).

[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.

[CF] M.A. Casanova and A.L. Furtado, "On the Description of Transition Constraints using Temporal Languages", in Advances in Database Theory, Vol. II, H. Gallaire, J. Minker and J-M. Nicolas (eds.), Plenum Press (to appear).

[DMW] W. Dosch, G. Mascari and M. Wirsing, "On the Algebraic Specification of Databases", Proc. 8th Int'l. Conf. on Very Large Data Bases (1982), 370-385.

[En] H.B. Enderton, A Mathematical Introduction to Logic, Academic Press (1972).

[FVC] A.L. Furtado, P.A.S. Veloso and M.A. Casanova, "A Grammatical Approach to Data Bases", Proc. 9th IFIP World Computer Congress (1983), 705-710.

[Ha] D. Harel, First-Order Dynamic Logic, LNCS Vol.68, Springer-Verlag (1979).

[He] W. Hesse, "A Correspondence Between W-Grammars and Formal Systems of Logic and its Application to Formal Language Description", Tech. Rep. TUM-INFO-7727, Technische Universität München (1977).

[VCF] P.A.S. Veloso, M.A. Casanova and A.L. Furtado, "Formal Data Base Specification - An Eclectic Perspective", Technical Report 1/84, Pontificia Universidade Catolica do RJ (1984).

[VF] P.A.S. Veloso and A.L. Furtado, "Stepwise Construction of Algebraic Specifications", in Advances in Database Theory Vol.II, H. Gallaire, J. Minker and J.-M. Nicolas (eds.), Plenum Press (to appear).