Entity-Relationship Approach to Software Engineering

PART -2-

005.106
E61
V.2

Entity-Relationship Approach
to Software Engineering

Entity-Relationship Approach to Software Engineering
C.G. Davis, S. Jajodia, P.A. Ng and R.T. Yeh (eds.)
Elsevier Science Publishers B.V. (North-Holland)

VIEW CONSTRUCTS FOR THE SPECIFICATION AND DESIGN OF EXTERNAL SCHEMAS

P.A.S. Veloso
A.L. Furtado

Departamento de Informática
Pontifícia Universidade Católica do Rio de Janeiro
22453 Rio de Janeiro RJ
Brasil

A view construct is proposed which consists of derived objects and operations defined on them. Attention is focussed on the update operations, for which the conditions, effects, and side effects must be specified. This construct is a template, appropriate both for abstract specification and concrete realization. A simple data base environment is described and used to illustrate the suggested approach, at three levels: informal description, formal specification and representation in terms of (a simplified version of) the entity-relationship model.

1. INTRODUCTION

End-users interact with data bases by posing questions relative to facts contained in the data base or by modifying such facts as a consequence of actions occurred in the real world. Thus a user interface to a data base can be characterized by the query and update operations that he is authorized to execute.

In the ANSI/X3/SPARC architecture [1], user interfaces correspond to external schemas. In most cases an external schema does not encompass the entire data base as described by its conceptual schema. In line with the considerations above, we characterize external schemas by the respective query and update operations. The mapping between an external schema and the conceptual schema is indicated, in turn, by the facts (or queries interrogating them) at the conceptual schema corresponding to or affected by the external schema operations.

From the algebraic approach to abstract data types we know, as a general principle, that updates and queries are mutually related in the sense that updates can affect the result of queries, whereas queries may constitute pre-conditions for the execution of updates. As shown in [13] updates can be defined in terms of pre-conditions and effects (on queries) so as to enforce the declared integrity constraints: the definition should contain for each update

a. pre-conditions for its execution;

b. the intended effect;

c. possible side-effects (i.e. effects not directly intended or even seen by the user invoking the update).

The name and parameters of updates constitute their syntax, whilst itens a, b, c above give their semantics. A third constituent should be added: an algorithm or program able to eventually lead to the execution of the update in a computer-based environment. This algorithm should work on some abstract structure if we want to stay at the very high level required at this stage. Such an abstract structure is provided by a suitable data model, which in our research has been the entity-

relationship [2] model or some variation thereof.

Proponents of abstract data types have been divided into the following two categories: those who specify a data type by way of axioms or equations [6,7] and those who make the concept useable by providing language constructs [8]. We contend that it is not the case to give preference to one or the other category since they are both useful and in fact complement each other.

In this paper we propose a language construct which takes both positions into consideration. The construct may serve as a paradigm for the design of appropriate constructs in specific languages supporting the entity-relationship model. We call it a view, in consonance with the usual terminology for the external schema level; however, as noted in [4], views should act not only as windows but also as shades and screens, hiding unauthorized information and disallowing illegal manipulations.

Views consist of virtual objects derived from actual objects pertaining to the conceptual schema. They should be designed by the enterprise administrator in consultation with the application administrators [1] in charge of the various external schemas.

In the next section a view construct is introduced and commented upon. Section 3 presents a simple example of a data base environment, which is used in section 4 to illustrate the design of views and their representation in (a simple version of) the entity-relationship model. Finally, section 5 contains some concluding remarks.

2. CONSTRUCTS FOR VIEWS

As indicated, a view consists essentially of derived objects together with derived operations on them. For instance, a view may consist of employees not currently assigned to projects together with the operations of hiring and firing. (This example will be developed in more detail in the following sections).

Accordingly, in order to describe a view we must specify how this derived sort is obtained from the basic ones and we must specify the derived operations. More formally, this amounts first to the construction of a virtual sort $S_1 \times \ldots \times S_n$ in terms of the primitive sorts $S_1,\ldots,S_n$ and then the selection of those objects that satisfy a property, called the view invariant. Thus, in set-theoretical notation, this virtual view sort may be described as

$$S = \{(s_1,\ldots,s_n) \in (S_1 \times \ldots \times S_n) / \rho(s_1,\ldots,s_n)\}$$

where $\rho(-,\ldots,-)$ is the view invariant. More generally, we might have a term t_i in lieu of s_i but we shall not employ this extra generality here.

As mentioned, views are to be designed by the enterprise administrator in consulation with the application administrators. Thus, a first step towards describing a view might use the following format

```
view NAME
  {comment describing the view}
  Displays
    objects names of the basic sorts involved
            {comment describing the view invariant}
    operations for each derived operation
            name (parameter list)
            {comment describing the operation}
end
```


The reader may at this point wish to refer to section 4 for a specific example.

The above format of view specification is oriented towards syntax. Indeed, only the syntactic aspects of the derived sorts and operations are formally described, the semantic ones being informally expressed by comments in natural language.

After designing all the views it is necessary to verify that the update operations indeed preserve the constraints, are mutually compatible, and are sufficient to handle the data base [11]. For this purpose a more formal specification of the views is more appropriate. Accordingly we propose a second format for the specification of a view as follows

```
view NAME
  Specification
    objects s1:S1,...,sn:Sn ; ρ(s1,...,sn)
    operations for each operation
      name ω(parameter list)
      conditions φ(s1,...,sn)[σ]
      effects ψ(s1,...,sn)[σ,τ]
      side effects θ(s1,...,sn)[σ,τ]
end
```

In the above the objects - part describes the virtual view sort using a stylized set-theoretical notation (reminiscent of the relational calculus [3]). For the operations - part some explanation is in order.

Let us assume that the current state of the data base is σ and operation ω is to be executed with parameters $s_1,\ldots,s_n$. The resulting state will be $\tau := \omega(s_1,\ldots,s_n)[\sigma]$. In the current state the view invariant must be satisfied by the parameters, i.e. $\rho(s_1,\ldots,s_n)[\sigma]$ holds. Now the conditions - part describes the precondition of the operation; if it fails the state will not be altered and we have $\tau=\sigma$. If, on the other hand, the precondition does hold then the operation will be successfully executed yielding a new state τ, which will be related to σ via the formulas in the effects and side effects parts of the specification. Thus, the specification given for the operation ω is equivalent to the following logical axiom

$$(\tau=\omega(s_1,\ldots,s_n)[\sigma] \wedge \rho(s_1,\ldots,s_n)[\sigma]) \rightarrow$$

$$\rightarrow [(\phi(s_1,\ldots,s_n)[\sigma] \rightarrow \psi(s_1,\ldots,s_n)[\sigma,\tau] \wedge \theta(s_1,\ldots,s_n)[\sigma,\tau]) \wedge$$

$$\wedge (\neg\phi(s_1,\ldots,s_n)[\sigma] \rightarrow \tau=\sigma)]$$

After the verification phase come the phases leading to an implementation. These phases do not require the participation of the above administrators, except that the implementors verify (and convince the administrators) that the views as finally programmed correspond to the design. (Also the data base administrator might be consulted with respect to the efficiency of the implementation.)

For these phases the implementors may receive the views specified in the second format above and proceed to represent them in a data model. Thus, the logical description of the derived objects and operations will be refined to one in terms of the primitive operations of the chosen model. Accordingly we have a third format for the view on this level, namely

```
view  NAME
   Representation
      objects expression, in the language of the model, describing how the objects
              of the view are obtained from the basic objects
      operations for each operation
          a program describing the operation in terms of primitive operations of
          the model
end
```

Notice that these three formats follow the same template. In the course of a process of top-down refinement it might be useful to consolidate some of these several formats. Of course for the user only the Displays and Specification formats should in principle be visible. Accordingly, the end-user sees

```
view  NAME
   Displays
      objects ...
      operations ...
   Specification
      objects ...
      operations ...
end visible part
```

This corresponds to the specification of an abstract data type in a query-oriented formalism [13].

On the other hand, the implementor would produce a documentation consisting of the Displays format together with the Representation format, perhaps followed by other levels of refinement, all of them, but the first one, hidden from the end-user. It would have the following aspect

```
view  NAME
   Displays
      objects ...
      operations ...
   Representation
      objects ...
      operations ...
end hidden part
```

The similarity with programming language constructs such as CLU's clusters [8,9] is apparent and not surprising as this corresponds to a representation of an abstract data type in terms of a more concrete one, the data model. (Accordingly protection can be achieved by requiring that a procedure call involve both the view and the operation name, thus NAME $ ω).

In a separate paper [12], a modular strategy for data base design, including the construction of views, is proposed. Theoretical aspects are investigated as well as how to pass from a specification to a representation on some data base management system.

3. A SIMPLE DATA BASE ENVIRONMENT

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

A. The basic sorts

The sorts treated in our example are:

N - name of employee
S - salary
J - job title
K - skill
T - task
P - project
L - leader of a project

The basic queries are:

emp(n,s,j) - employee's name, salary and job title
req(t,k) - requirement of a skill to do a task
assn(n,t,p)- assignment of an employee in a project to a task
mng(p,l) - management of a project by a leader
cap(n,k) - capabilities (skills) possessed or acquired by employees

Figure 1 indicates the arities of these queries, regarded as predicates, leaving implicit an argument sort, the data base state

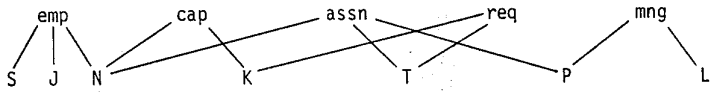

Figure 1
Basic queries and sorts

B. The users

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

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

Query emp does not give information about these special employees.

C. The activities of the enterprise

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

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

Various tasks compose each project, and the engineering manager is responsible for indicating what skills are required of an employee to perform each task. The engineering manager also associates employees with projects (but not suspended ones), and terminates such associations.

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

Each leader of a project determines how employees associated with his project are

assigned to tasks. An employee must possess all the skills required for each task assigned to him.

D. Constraints

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

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

Notice that constraints 1 through 7 are static constraints. They refer only to the current data base state, which we leave implicit, for legibility, in the following formalization. Also implicit are leading universal quantifiers, as usual.

1. $emp(n,s,j) \rightarrow s \geq min$
2. $emp(n,s,j) \wedge emp(n,s',j') \rightarrow s=s' \wedge j=j'$
3. $assn(n,t,p) \rightarrow (\exists s:S)(\exists j:J)\ emp(n,s,j)$
4. $cap(n,k) \rightarrow (\exists s:S)(\exists j:J)\ emp(n,s,j)$
5. $mng(p,l) \wedge mng(p,l') \rightarrow l=l'$
6. $assn(n,t,p) \rightarrow (\exists l:L)\ mng(p,l)$
7. $assn(n,t,p) \rightarrow (\forall k:K)(req(t,k) \rightarrow cap(n,k))$

On the other hand, constraints 8, 9 and 10 are transition constraints, in that they involve both the current data base state σ and the next state τ resulting from the application of an update. A possible formalization for them is as follows

8. $\tau=initiate(p,l)[\sigma] \rightarrow mng(p,l)[\tau]$
9. $\tau=terminate(p)[\sigma] \rightarrow$
   $\rightarrow((\exists l:L)mng(p,l)[\sigma] \rightarrow \tau=\sigma)$
10. $\tau=fire(n)[\sigma] \rightarrow$
    $\rightarrow((\exists t:T)(\exists p:P)assn(n,t,p)[\sigma] \rightarrow \tau=\sigma)$

4. DESIGN OF THE EXTERNAL SCHEMAS

We shall now outline the design of the external schemas of the recognized users of the simple example in the preceding section. We give only some views in detail in order to illustrate the constructs. However, as argued before, the design of the views is interdependent. Thus, for each user, we shall exhibit his schema by means of a diagram and a short explanation of his views. In these diagrams we adopt the convention that dotted lines lead to sorts that can only be interrogated but not updated.

A. Personnel Manager's Schema

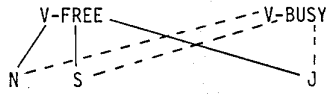

Figure 2:
The personnel manager's schema

V-FREE(N,S,J) - employees not associated with projects
V-BUSY(N,S,J) - employees associated with at least one project

The visible part of the view V-FREE is as follows

```
view V-FREE
  {employees not associated with projects}
    Displays
      objects N,S,T {such that n:N does not appear in assn}
      operations
           hire(n,s,j) {hire employee with name n, salary s and job title j}
           fire(n) {fire employee n}
    Specification
      objects(n:N,s:S,j:J);(∀t:T)(∀p:P) ¬assn(n,t,p)
      operations
       name  hire(n,s,j)
        conditions  ¬(∃s':S)(∃j':J)emp(n,s',j')[σ]
                    s ≥ min
        effects emp(n,s,j)[τ]
        side effects -
       end  hire
       name  fire(n)
        conditions (∃s':S)(∃j':J)emp(n,s',j')[σ]
        effects(∀s':S)(∀j':J) ¬emp(n,s',j')[τ]
        side effects (∀k:K) ¬cap(n,k)[τ]
       end  fire
end visible part
```

The view V-BUSY has no update operations. So its visible part is simplified to

```
view V-BUSY
  {employees associated with projects}
    Displays
      objectsN,S,T {such that n:N does appear in assn}
    Specification
      objects(n:N,s:S,j:J);(∃t:T)(∃p:P)assn(n,t,p)
end visible part
```

Following [14], we decided not to encapsulate query operations; we allow the views to inherit the basic queries. That is why they do not appear explicitly in the views. In view of this all operations in a view are updates with target sort data base, which we leave implicit, as we omit the argument sort data state in the name of the operations.

B. Engineering Manager's Schema

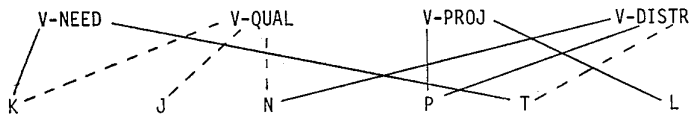

Figure 3:
The engineering manager's schema

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

The visible parts of the first two views are as follows.

```
view V-NEED
   {skills required for tasks}
      Displays
         objects T,K
         operations
              require(t,k) {makes skill k required for task t}
              remove(t,k) {makes skill k no longer required for task t}
      Specification
         objects (t:T,k:K)
         operations
           name require(t,k)
            conditions ¬ req(t,k)[σ]
            effects req(t,k)[τ]
            side effects (∀n:N)(∀p:P)(¬ cap(n,k)[σ] →
                                        →¬ assn(n,t,p)[τ])
           end require
           name remove(t,k)
            conditions req(t,k)[σ]
            effects ¬ req(t,k)[τ]
            side effects -
           end remove
end visible part

view V-QUAL
   {qualifications of employees}
      Displays
         objects N,J,K {such that (n,k):N×K appears in cap and
                                  (n,j):N×K appears in emp}
      Specification
         objects (n:N,j:J,k:K);cap(n,k) ∧ (∃s:S)emp(n,s,j)
end visible part
```

For the other two views their Displays-parts are as follows.

```
view V-PROJ
   {projects and their leaders}
      Displays
         objects P,L
         operations
              initiate(p,l) {initiate project p with leader l}
              replace(p,l) {replace current leader of project p by l}
              suspend(p) {suspend project p by removing its leader}
              restart(p,l) {restart project p by assigning l as leader}
              terminate(p) {terminate project p, if possible}
end

view V-DISTR
   {distribution of employees to tasks in projects}
      Displays
         objects N,T,P {such that n:N is in emp}
         operations
              associate(n,p) {associate employee with name n to project p,
                              if possible}
              disassociate(n,p) {disassociate employee n from project p}
end
```


C. Training Manager's Schema

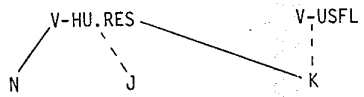

Figure 4:
The training manager's schema

V-HU.RES(N,J,K) - human resources
V-USFL(K) - useful skills, i.e. skills required by at least one task

The Displays - parts of these views are as follows.

```
view V-HU.RES
   {skills possessed by employees}
      Displays
         objects N,J,K {with (n,j):N×J from emp}
         operations
              acquire(n,k) {employee n acquires skill k}
              lose(n,k) {employee n loses skill k}
end

view V-USFL
   {skills required for some task}
      Displays
         objects K {from req}
end
```

D. Project Leader's Schemas

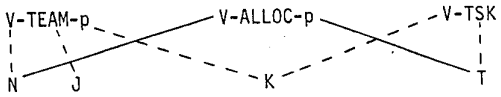

Figure 5:
The external schema of the leader of each project p

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

The Displays - parts of these views are as follows.

```
view V-TEAM-p
   {employees assigned to project p}
      Displays
         objects N,J,K {from emp and cap such that for some t:T (n,t,p) is in assn}
end

view V-ALLOC-p
   {allocation of employees to tasks in project p}
      Displays
         objects N,T {from emp}
         operations
               assign(n,t) {assign employee n to task t in project p, if  n
                            has all the skills required}
               release(n,t) {release employee n from task t in project p}
end

view V-TSK
   {skills required for a task}
      Displays
         objects T,K {from req}
end
```

E. Representation

In order to present the views on the representation level we must choose a data model. Here we decided to employ the entity-relatioship data model [2]. Our simple example can be modelled by the diagram using Chen's conventions in Figure 6.

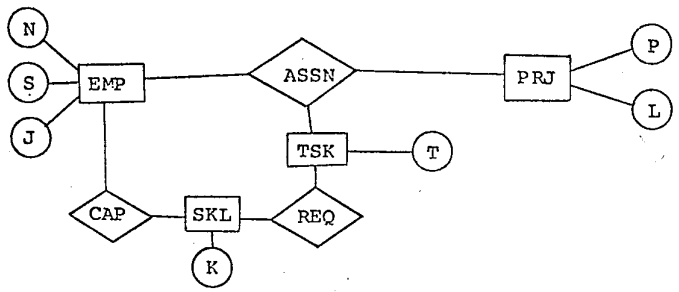

Figure 6:
Entity-relationship diagram for the example

We represent employees, tasks, projects and skills as entities with corresponding entity-sets EMP, TSK, PRJ and SKL. The attributes are, as indicated in the diagram, NAM, SAL and JTL for an employee, PID and LDR for projects, TID for tasks and SID for skills. In addition we have the relatioship sets CAP and REQ, both consisting of binary relationships, and ASSN, consisting of ternary relationships.

Thus, we do not need the full ER model for this particular example. Rather, for simplicity sake, we shall confine ourselves to a restricted version of the ER model, supporting only binary and ternary relationships and allowing attributes for entities but not for relationships.

This simplified model is a version of the S-ER data model [5]. Its primitive update operations permit to initialize (phi) the data base to an empty state, create (crt) and delete (del) entities within entity-sets, modify (mod) values of attributes ('*' standing for the undefined value) and link ($lk2$,$lk3$) or unlink ($ul2$,$ul3$) entities via a (binary or ternary) relationship. Its primitive query operations are predicates referring to the existence (exs) of entities within entity-sets, values ($hv\ell$) of attributes and relatedness ($r\ell2$,$r\ell3$)of entities. Similar operations were introduced in [10] whereas [5] contains a formal specification of a version of the S-ER data model.

On the representation level, the operations of a view are described by a program. Of course, our construct is not tied down to any specific programming language and we employ here only self-explanatory features.

The representation of the views of the personnel manager's schema are as follows.

```
view V-FREE
    Representation
       objects e:ent; (∀t:ent)(∀p:ent)¬rℓ3(e,t,p,ASSN)
       operations
          name hire(n:N,s:S,j:J)
          body if s <min ν(∃e:ent)(exs(e,EMP)[σ] ∧ hvℓ(e,NAM,n)[σ])
                  then return σ
                  else return mod(e,NAM,n)[mod(e,SAL,s)[
                                           mod(e,JTL,j) crt(e,EMP)[σ]]]]
          end hire
          name fire(n:N)
          body if ¬(∃e:ent)(exs(e,EMP)[σ] ∧ hvℓ(e,NAM,n)[σ])
                  then return σ
                  else begin
                       let e:ent be such that exs(e,EMP)[σ] ∧
                                              ∧ hvℓ(e,NAM,n)[σ];
                       let τ:=mod(e,NAM,*)[mod(e,SAL,*)[
                                 mod(e,JTL,*)[σ]]];
                       for all k:K such that rℓ2(e,k,CAP)[τ]
                            do τ:=uℓ2(e,k,CAP)[τ];
                       return del(e,EMP)[τ]
                       end
          end fire
end V-FREE

view V-BUSY
    Representation
       objects e:ent; (∃t:ent)(∃p:P)rℓ3(e,t,p,ASSN)
end V-BUSY
```

The above descriptions were obtained by translating the corresponding specification into our simplified ER model. Notice that the body of each operation ω has the following general pattern

```
if ¬φ{precondition for ω fails}
   then return σ
   else {φ ∧ ρ}program for ω {ψ ∧ θ}
```

We can proceed similarly with the other views. We have decided not to encapsulate the query operations, but they should also be described in terms of the primitive operations of the data model. In our example this is quite straightforward, for instance

$$emp(n,s,j) \leftrightarrow (\exists e{:}ent)(exs(e,EMP) \wedge hvl(e,NAM,n) \wedge hvl(e,SAL,s) \wedge hvl(e,JTL,j))$$
$$cap(n,k) \leftrightarrow (\exists e,f{:}ent)(exs(e,EMP) \wedge exs(f,SKL) \wedge hvl(e,NAM,n) \wedge hvl(f,SID,k) \wedge rel(e,f,CAP))$$

Then we can employ any implementation of this data model.

5. CONCLUSIONS

We have proposed a construct for the specification and design of views. This construct embodies the two main approaches to abstract data types, namely implicit specification and programming language realization. It provides a paradigm for embedding specific versions of the constructs into languages supporting, for instance, the entity-relationship model.

Here we have favored the decision of directly translating the views in terms of the data model. An alternative, deserving further attention, would employ modules; a view module, specifying the view in terms of the conceptual schema, and a representation module, representing the conceptual schema abstract data type in terms of a data model. This alternative would lead to a modular data architecture as in [5].

REFERENCES

[1] ANSI/X3/SPARC - Interim Report of the Study Group on Data Management Systems, FDT Bulletin, ACM (1975).

[2] Chen,P. The entity-relationship model: toward a unified view of data, ACM TODS,1(1976) 9-36.

[3] Codd,E.F. Relational completeness of data base sublanguages, in: Rustin,R. (ed.) Data Base Systems (Prentice-Hall, 1972) 65-98.

[4] Furtado,A.L.; Sevcik,K.C.; Santos,C.S. Permitting updates through views of data bases, Information Systems 4 (1979) 269-283.

[5] Furtado,A.L.; Veloso,P.A.S.; Castilho,J.M.V. Verification and testing of S-ER representations, in: Chen,P. (ed.) Entity-Relationship Approach to Information Modelling and Analysis (ER Institute, 1981) 125-149.

[6] Goguen,J.A.; Thatcher,J.W.; Wagner,E.G. An initial algebra approach to the specification, correctness and implementation of abstract data types, in: Yeh,R.T. (ed.) Current Trends in Programming Methodology, vol.IV (Prentice-Hall, 1978) 81-149.

[7] Guttag,J. Abstract data types and the development of data structures, Comm. ACM, 20 (1977) 396-404.

[8] Liskov,B.H.; Snyder,A.; Atkinson,R.; Schaffert,C. Abstraction mechanisms in CLU, Comm. ACM, 20 (1977).

[9] Liskov,B. et al. CLU Reference Manual (Springer-Verlag, 1981).

[10] Santos,C.S.; Neuhold,E.J.; Furtado,A.L. A data type approach to the entity-relationship model, in: Chen,P. (ed.) Entity-Relationship Approach to Systems Analysis and Design (North-Holland, 1980).

[11] Sevcik,K.C.; Furtado,A.L. Complete and compatible sets of update operations, International Conf. on Management of Data (Milan, 1978).

[12] Tucherman,L.; Furtado,A.L.; Casanova,M.A. A pragmatical approach to structured database design, Proc. 9th Conf. on Very Large Data Bases (Florence, 1983).

[13] Veloso,P.A.S.; Castilho,J.M.V.; Furtado,A.L. Systematic derivation of complementary specifications, Proc. 7th Conf. on Very Large Data Bases (Cannes, 1981).

[14] Zilles,S.N. Types, algebras and modelling, SIGMOD Record, 11 (1981) 207-209.