INFORMATION SYSTEM ENGINEERING : THE RUBIS SYSTEM

C. CAUVET *, C. ROLLAND *, J.Y. LINGAT **

* Université Paris 1 UFR 06
17, Rue de la Sorbonne
75231 PARIS Cedex 5 FRANCE

** THOM'6 33, rue Vouille
75015 PARIS FRANCE

ABSTRACT:

The paper aims to present a CASE system which is an integrated computer aided tool so-called RUBIS, for designing and prototyping Information Systems.
RUBIS is organized around the R-schema (RUBIS-schema), which is a declarative specification of the Information System (IS) content, based on a conceptual model which emphasizes equally the IS structure and behaviour.
After a brief overview of the RUBIS architecture, we present through examples how to construct the R-schema using PROQUEL: the formal specification language of RUBIS. Then, we concentrate on two different aspects of RUBIS: the expert design tool, which helps the designer to produce the R-schema for a given application domain, and the prototyping tool, which allows the execution of specifications on test case data.

1. INTRODUCTION

The paper aims to present an integrated computer aided tool so-called RUBIS [33], for designing and prototyping Information Systems (IS).

RUBIS works on a **R-schema** (RUBIS-schema) which is a declarative specification of the IS, made using the formal specification language called PROQUEL (PROgramming QUEry Language) [34]. The R-schema provides a conceptual description of both the static and dynamic aspects of the IS to be built. It is based on the REMORA model [1]. The static IS aspects are modelled by **relations** while **operations** (elementary actions on an object) and **events** (elementary state changes triggering one or several operations) allow the modelling of the dynamic aspects of objects. Thus, the R-schema is a collection of relations, operations and events specifications. In this schema the temporal aspects of the application are also taken into account; they are modelled by using the time types and functions provided by the RUBIS Time Model.

RUBIS includes **three different interfaces** as computer aided design tools:

- a **Graphic Interface** using an icon-based representation of the R-schema concepts.

- an **Expert Design Tool** working on a semantic network, and helping the designer to create, validate and improve the conceptual schema.

- a **Menu Interface** based on the PROQUEL language.

The Graphic and Menu based interfaces are both devoted to experienced designers while the Expert Design Interface can be used by relatively unexperienced analysts or designers.

Prototyping in RUBIS is achieved by the Temporal Processor which manages temporal aspects of the specification and by the Event Processor which manages event recognition and synchronization. Both Processors use the PROQUEL interpreter.

Thus the PROQUEL Interpreter, the Event Processor and the Temporal Processor are the key tools for prototyping information systems with RUBIS. They have been implemented as extensions of a Relational DBMS [2].

The paper is organized as follows:

The global architecture of RUBIS is given in section 2 with a brief description of the different RUBIS system modules.
Section 3 presents how to specify the R-schema using the PROQUEL language. Then, the paper focusses in section 4 on one of the three interfaces, namely the Expert Design Interface. The Event Processor is detailed in section 5.

2. RUBIS ARCHITECTURE

The architecture of the RUBIS system is presented in figure 2.1. This displays the three major aspects of the system, which are:

1. The (meta) data mananagement tools which handle the prototype database and the specification database (containing the R - schema).

2. The R-Schema design interfaces and the Validation Module.

3. The prototyping tools - the Application Monitor, the Event Processor and the Temporal Processor.

Each of these three aspects is introduced in turn.

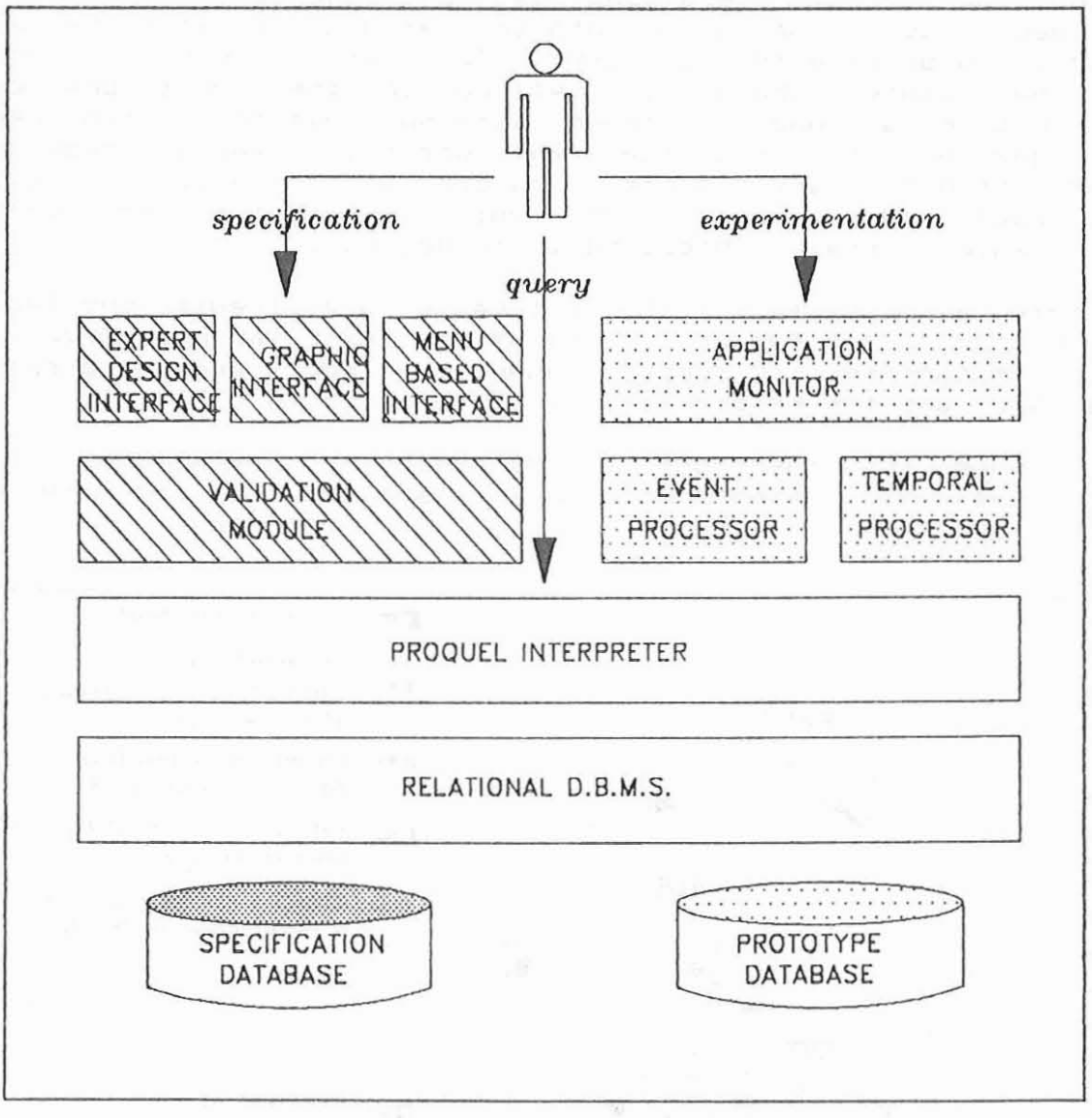


Figure 2.1 : Architecture of the RUBIS system

2.1 The R-schema and the specification database

The R-schema is a modular description of the conceptual schema for the Information System being developed. This schema is based on the model of the REMORA methodology [1] [3], and describes both static aspects (structure) and dynamic aspects (behaviour) of the IS.

The static aspects are modeled using **relations** representing entities or entity associations in the real world (e.g. client, invoice, loan, etc.).

The dynamic aspects are modeled using:

* **Operations** which represent elementary actions on an application object (e.g. add a new client, modify an order, etc.).

* **Events** which represent elementary state changes in the system at which time certain operations must be triggered (e.g. when an order arrives, insert the order into the database, reserve the requested goods, and prepare for delivery). The description of the conditions for the state change is defined in the event predicate. A distinction is made between external events (which represent messages received from the real world), internal events (which represent elementary state changes of a relation within the database), and temporal events (which represent temporal conditions under which certain processing is triggered).

The temporal aspects of the application are likewise modeled, using the functions and temporal types of the RUBIS Temporal Model. Due to space limitations, the temporal aspects of RUBIs are not presented in this paper and the reader is invited to refer to [32].

The R-schema is therefore a collection of relations, events and operations. The content of the R-schema can be illustrated using a graph (fig. 2.2)

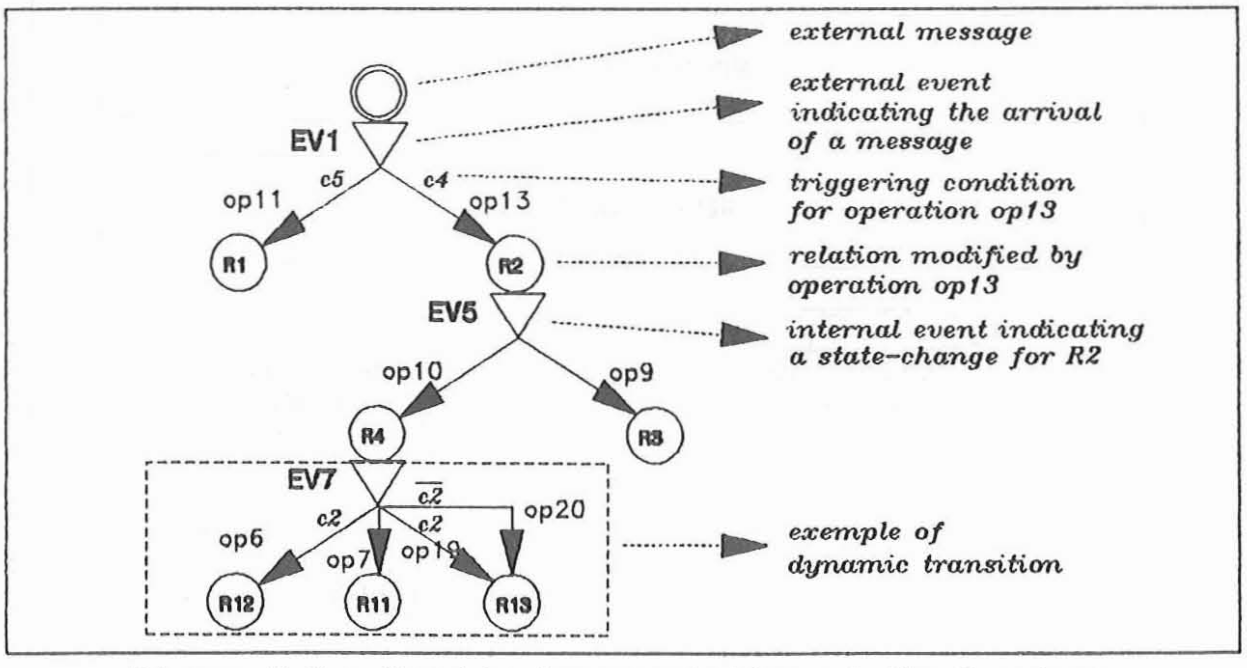


Figure 2.2 : Graphic Representation of the R-schema

Such a representation introduces the dynamic transitions of the IS, showing their sequence and precedence. A dynamic transition is composed of (1) an event (2) all the operations triggered by the event (3) all the relations modified by these operations. This corresponds to an elementary database transaction, since by definition a RUBIS transition is atomic, and must pass the database from one coherent state to another. Notice that the triggering of an operation by an event can be conditionnal (using a triggering condition) and iterative (using a triggering factor)

THE META-BASE

The R-schema is stored into the specification database, also called meta-base. The IS specifications are called metadata to distinguish the context of the meta-base in a relational form defined by a meta-schema.
It is extremely important, during conceptual design, for the designer to get support from the RUBIS system to access to the meta-base. This support is provided both by the three computer based interfaces which allow the designer to insert, modify and delete meta-data. The PROQUEL language allows to directly interact with the tuples of the meta-base and is used by the three design interfaces. For example, the designer can modify the specification text of an event predicate or an operation. Notice that such modification doesn't imply recompiling the application; it does not even imply stopping the user's activities if the text is not used at the moment.

In addition, the meta-base contains major project management informations (e.g history of R-schema elements, sessions and designers references ...).

Finally the meta-base is used to store informations used by the system itself during the prototyping phase.
All the meta-data are stored in a relational form and are accessible through the PROQUEL query language.

2.2 The Aided Design Tools

* The **Graphic Interface** generates the R-schema from a graphic expression of both the static relations schemes and the dynamic transition graphs.
The tool is a user-friendly interface based on windows and icons for displaying information and on pop-up menus, keyboard and mouse for entering data. It provides graphical facilities for changing the drawings of graphical representations. In addition it allows the designer to simultaneously see on the screen several graphical descriptions of the R-schema.

* The scope of the **Expert Design Tool** (EDT) is to provide the designer with an active and intelligent support during the IS design process, leading to the R-schema.

The EDT is intended to behave like an expert designer using its own knowledge base, which is partly formal and partly composed of experimental design rules.

The EDT starts with an Object Oriented description of the application domain and progressively helps the designer to correct, complete and make this description coherent before generating automatically the R-schema.

* The **Menu Based Interface** provides the designer with a guided interaction to enter the R-specifications into the meta-base. Insertion, modification, deletion of metadata are driven by sequences of menus, thus decreasing the designer effort in specifying the meta-data to be inserted, modified or deleted from the meta-base.

* The **Consistency checking module** aims at analysing the meta-base and detecting the presence of undesirable features i.e specifications not satisfying general design criteria. Checks are of three types:

- **correctness** checks verify that all R-schema elements are correct with respect to the RUBIS modeling concepts and with the PROQUEL syntax.

- **completeness** checks detect missing elements in the R-schema.

- **accuracy** checks are the most sophisticated checks. They detect possibly inconsistencies in the R-schema and interact with the designer in order to decide if corrections are needed or not.

Consistency checks are performed on the specifications at various moments of the conceptual design process. Part of the checks are included in the designer interfaces. Global consistency checks are automatically performed upon completion of one conceptual design session. They can be initiated by the designer at any moment either on the global content of the meta-base or on a subset.

2.3. The prototyping tools

* The **application monitor** is the end-user interface. For each external event specification, a corresponding Application Program (AP) is generated. The AP construction is based upon the event structure . The application monitor executes Application Programs according to end-user requests. In fact, executing an AP corresponds to a message acquisition and validation. When the AP is correctly finished, the application monitor sends the valid message into the message queue of the Event Processor.

Since the external event predicate is verified by the corresponding AP, one may consider the reception of a valid message in the Message Queue as an external event occurrence. Each time a user is connected to RUBIS, a process containing an application monitor is created.

* The **Temporal processor** works independently. It sends a message into the Message Queue each time it recognizes a temporal event. The Temporal Processor is fully described in [32].

* The **Event processor** recognizes internal events, takes into account external and temporal events, processes and synchronizes all events. The event Processor is detailed in section 5.

* The **PROQUEL interpreter** executes all texts written in PROQUEL, by sending queries to the DBMS and managing local variables, control structures and parameter passing. It has been developed using LEX and YACC tools of the UNIX system. Queries (expressed in relationnal algebra) are send to a small Relationnal DBMS called PEPIN [2].

3. SPECIFYING THE R-SCHEMA

In this section, we detail the R-schema specification using the PROQUEL language. First, we can remark that the description of the R-schema can be made incrementally:

- first, the static sub-schema can be described with relation specifications (introduced by DEFINE RELATION),

- second, a first version of the dynamic sub-schema can be obtained by specifying dynamic transitions (these specifications are introduced by DEFINE EVENT),

- third, the dynamic sub-schema can be completed by operation, condition and factor specifications (respectively introduced by DEFINE OPERATION, DEFINE CONDITION and DEFINE FACTOR).

We illustrate this process by considering the framework for an automated subscription-library management system. The examples introduced center around the following relational schema:

```
BOOK (BOOK#, PUBLISHER, TITLE)
COPY (BOOK#, COPY#, ACQDATE, PRICE, CP_STATUS)
SUBSCRIBER (SUBSC#,NAME,ADDRESS,SUBDATE,SUBSC_STATUS,NUMCOPIES)
REQUEST (REQ#, SUBSC#, REQDATE, REQTYPE, BOOK#, REQ_STATUS)
LOAN (LOAN#, LOANDATE, BOOK#, COPY#, REQ#)
NOTICE (NTC#, NTC_DATE, LOAN#, SUBSC#)
```

The meaning of the attributes (when not obvious) will be given in the examples.

Figure 3.1 associates: (1) the current loan agreement in force at the library, (2) the graphical representation of the corresponding dynamic transition, and (3) the PROQUEL specification.
The specification is a translation into the PROQUEL language of the dynamic schema, which is itself a natural model of the loan agreement. The specification of the external event "loan request arrival" is composed of three parts:

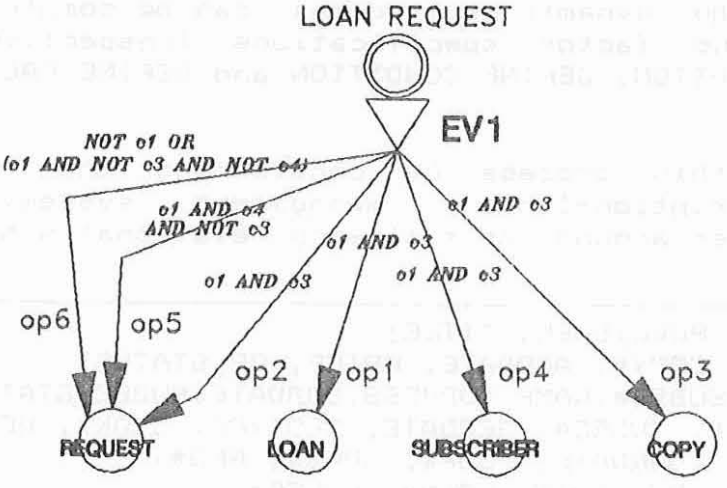
* The message "LOAN REQUEST" is described in PROP. It consists of a request number, a book number, the subscriber number, and the request type (immediate, or hold).
* The event predicate is specified in PRED. In the example, the "LOAN REQUEST" message is acceptable only if both the requested book and the subscriber are present in the database.
* The TRIGGER specifies the result of the arrival of the loan request. Three cases are possible:

1) The request can be satisfied (c3 : a copy of the book is available), and the subscriber is in good standing (c1 : his subscription is up to date, he has no outstanding late notice, and he has less than three books currently on loan). In this case, a loan is created (op1), the request is accepted (op2), the status for the book is set to "ON LOAN" (op3), and the number of copies on loan for this subscriber is incremented (op4).

TEXT SPECIFICATION

* a subscriber may not borrow more than three books simultaneously;
* a loan request is receivable if the subscriber is valid (his subscription is up to date and he has no overdue books);
* if a request is type "hold", the unavailability of the requested book causes the request to be put on hold.

GRAPHIC SPECIFICATION



PROQUEL SPECIFICATION

```
DEFINE EVENT ev1 IS request_arrival
  ON MESSAGE
  COMMENT "Arrival of a loan request"
  PROP { num_req : INTEGER;
         num_book : INTEGER;
         num_subsc : INTEGER;
         type : (IMMEDIATE, HOLD); }
  PRED { (EXISTS book WHERE book# = CONTEXT.num_book)
          AND (EXISTS subscriber WHERE subsc# = CONTEXT.num_subsc)}
  TRIGGER { IF c1 AND c3 THEN { op1 ON loan;
                                op2 ON request;
                                op3 ON copy;
                                op4 ON subscriber; };
            IF c1 AND NOT c3 AND c4 THEN op5 ON request;
            IF NOT c1 OR (c1 AND NOT c3 AND NOT c4)
                 THEN op6 ON request; };
```

Figure 3.1 : Dynamic Transition Specification

2) The request cannot be satisfied (there is no available copy), but the subscriber is in good standing and wishes to leave his request on hold. The demand is put on hold (op5).

3) The subscriber is not in good standing, or the request cannot be satisfied for an immediate request. In this case, the request is refused, but is still added to the database for statistical purposes (op6).

The specification of an event in PROQUEL defines the structure of the associated dynamic transition. The elements defined in the TRIGGER (conditions and operations) are defined separately. This allows a progressive and modular description of the application. In addition, the same condition or operation may be shared by several dynamic transitions.

For example, figures 3.2 and 3.3 show the specification of condition c1 and of operation op1.

```
DEFINE CONDITION c1 IS good_standing
COMMENT "The subscription is up to date, no pending late notices,
         and less than three books on loan"
TEXT ( VAR $status : STRING;
       VAR $ncopy : INTEGER;
       VAR $nolate : BOOLEAN;
       SELECT UNIQUE subsc_status, numcopies INTO $status, $ncopy
         FROM subscriber WHERE subsc# = CONTEXT.num_subsc;
       $nolate := NOT EXISTS notice
                    WHERE subsc# = CONTEXT.num_subsc;
       RETURN ( $status="VALIDE" AND $ncopy<3 AND $nolate ) );
```

Figure 3.2 : Specification of a Trigger Condition

```
DEFINE OPERATION op1 IS ins_loan
  COMMENT "Create loan"
  TYPE insert IN loan
  INPUT () (* no explicit parameters *)
  TEXT ( VAR $copy, $max : INTEGER;
         $copy := SELECT FIRST copy# FROM copy
                    WHERE book# = CONTEXT.num_book
                      AND cp_status = "AVAILABLE";
         $max := SELECT UNIQUE MAX(loan#) FROM loan;
         INSERT INTO loan ($max+1, current_date,
                           CONTEXT.num_book, $copy, CONTEXT.num_req);
       );
```

Figure 3.3 : Specification of an Operation

Each specification is a module independent of the others. The variables declared in a module have scope within that module. All modules receive an implicit call parameter referenced by the keyword **CONTEXT**. This parameter designates the message/tuple for which the arrival/state-change generated the event. In the specifications of ev1, op1 and c1, CONTEXT represents the loan request message.

When a distinction is necessary between the old and new value of the context (in the case of an internal event, for instance), the prefixes OLD and NEW are used.
The "context" of a dynamic transition is in fact an implicit parameter passed to all PROQUEL texts which it contains. Not having to declare each passage of the context by parameter clearly simplifies the developer's specifications.

A condition corresponds to a boolean function. Its text is terminated by the RETURN statement, which determines the return value of the condition.

Each operation possesses an implicit exit parameter : the two successive values of the tuple it modifies. If the state change generated by the execution of the operation generates an internal event, this parameter serves as the "context" for the dynamic transition of the generated event.

Figure 3.3 illustrates the contribution of the combined usage of variables and SQL queries. In the usual application development environment, the developer must resort to a language using "embedded SQL", with all the awkwardness this implies.

The meta-base is incremented progressively with the arrival of new specifications. The relational mapping of the R-schema is thus automatic, and the meta-base can be manipulated using the PROQUEL query language. The designer can also use the programming aspects of PROQUEL to construct his own procedures and tools for often used queries, validation checks on the meta-base, and so on. In RUBIS, PROQUEL constitutes one of the main factors of integration.

4. THE EXPERT DESIGN TOOL

4.1. Why an Expert Design Tool ?

The design of large IS is a complex, iterative, long and tedious task that must be supported by tools. The idea of computer aided design tools is not new [22], [23], [24].

Such tools provide help to memorize the results of the modelling activities, to check consistency and completness, to produce documentation and inform the design team on the status of design. They help in the management of schemas, but little in their production. Recent researches have stressed the models used for building schemas and the languages used to specifying them. But none progress has been made on the process of producing these schemas.

We believe that a step forward in the design of IS requires an effort of formalization of the intellectual process of construction of schemas in order to build software tool which brings a more effective help in design.

We are conscious of the fact that design is a non algorithmic activity. It is a complex task which is not very formal and is full of incertainity. The experienced designer masters the task of design because he uses, on the one hand, his formal knowledge of the models,

and on the other hand, his experience which enables him to recognize typical cases and to treat them by analogy, to be attentive to certain delicate aspects...

The nature of the task of design, therefore requires an effort of formalization of, on the one hand the algorithmic part of the design (for example algorithm of normalization) and on the other, the heuristic part (experimental rules of designers).

In addition, if we wish to support the design process by a tool, it must be able to include both formal knowledge and experimental knowledge.

For all these reasons an expert system approach seems appropriate.

This approach will allow us to reproduce the expert's attitude, to exploit experimental knowledge necessary to the mastering of design by combining it with more formal knowledge.

Our hypothesis was to define an expert system for aid to the design process. The help supplied by the tool relies on a knowledge base where the concepts and the formal rules governing their use are grouped together with the rules of experimental know-how of design experts. The quality of the expert system depends on the richness of its knowledge base.

To define this knowledge base we have sought to analyse the reasoning processes carried out by the designer during the design process in order to reproduce them in the expert system.

4.2 Expert design tool architecture and functions

In order to help designers in the R-schema production, we have chosen to develop a tool which allows them to start with a semantic view of the application domain, which provides guidance and tutoring to improve and detail it and finally maps it onto elements of the R-schema.

The overall architecture of the expert design tool (EDT) is illustrated on figure 4.1.

It shows that the EDT is organized around a semantic network which represents the conceptual schema content at the different steps of It shows that the EDT is organized around a semantic network which represents the conceptual schema content at the different steps of its design. It initially corresponds to a rough semantic view of the application domain, progressively corrected, completed and detailed before it can be mapped onto relations, events and operations of the R-schema.

The manipulation module allows the designer to create, modify or delete elements of the semantic network.

Using the query module, the designer can be inform on the current version of the conceptual schema.

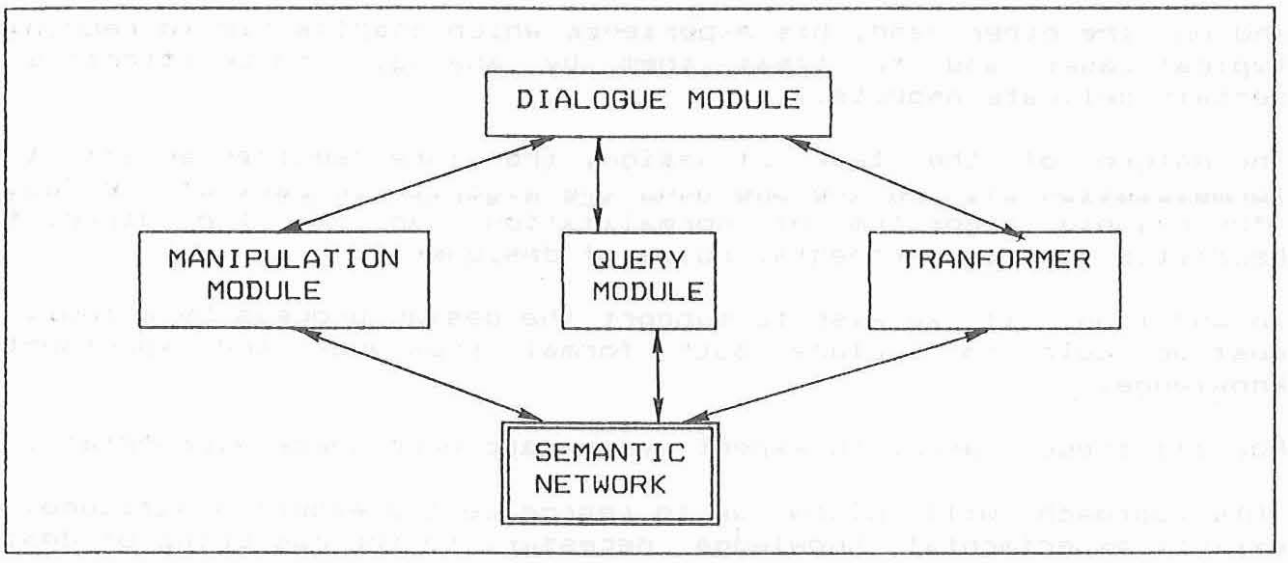


Figure 4.1 : Expert design tool architecture

The Semantic Network Transformer (SNT) is the most intelligent part of the interface. The SNT is intended to behave like an expert designer. Using its own design knowledge, it detects inconsistencies, incorrectness and incompletnesses, infers decisions and provides the designer with alternative solutions for improving the current version of the conceptual schema.

The SNT is organized as an expert system with a knowledge base and an inference engine. The knowledge base transformer includes a fact base (which is the semantic network) and a rule base including diagnosis rules, mapping rules and improvement rules. As the interface is implemented with PROLOG, the inference engine is the prolog interpreter.

We focus in the following on the SNT. We introduce in turn the semantic network and the transformer knowledge base.

4.3 The semantic network

The semantic network is an oriented and labelled graph. It comprises 5 types of node and 3 types of edge. Nodes and edges can be labelled.

The 5 types of node correspond to the 5 predefined objects on which we suggest designers to concentrate on.

The 3 types of edge represent the 3 predefined associations among objects we propose to investigate during the design process.

The two main design principles underlying the semantic network architecture are the following:

a) reality can be easily percieved in a causal way as illustrated on figure 4.2.

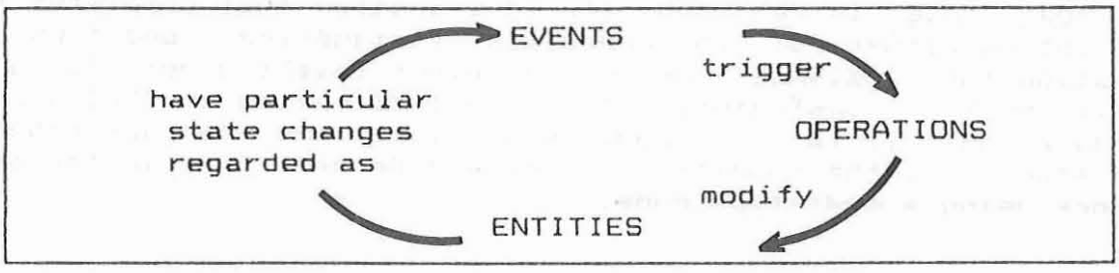


Figure 4.2

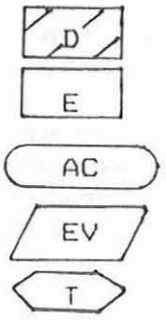
Event, Operation and Entity are 3 predefined types of node of the semantic network. We added domain and text types node in order to deal with properties of event, operation and entity.

b) reality is composed of complex objects. A way to deal with the complexity of complex objects is to use aggregation, generalization and grouping abstraction forms. They correspond to the 3 types of edge of the semantic network.

Aggregation, generalization and grouping apply on event, operation, domain and entity nodes. This means that the semantic view of an application domain we propose to describe through the semantic network is a hierarchy of complex events, operations, domains and entities.

Let us detail and examplify the semantic network types of node and types of edge using the following graphical notation.

THE NODES



domain node

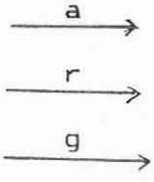
entity node

operation node

event node

text node

THE EDGES



aggregation edge

grouping edge

generalization edge

4.3.1 Semantic meaning of nodes

- A domain node represents a data type. It is used for the representation of entity properties, operation parameters and event contexts. A book number, a subscriber name are examples of domains.

- An entity node modelizes an entity type of the application domain, such as a book, a request, a subscriber...

- An operation node represents an action type. The action of a given type modify entities belonging to the same type. For example a request analysis, a new book insertion are described in the semantic network with operation nodes.

- An event node describes state changes of the real world that trigger executions of similar operations. A book is just arrived, an instance of book becomes available, are examples of event types.

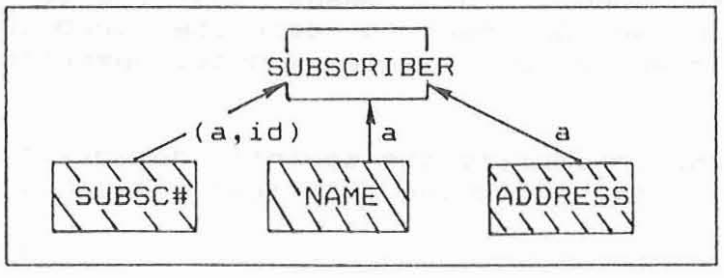
- A text node is a predicate declaration that completes the description either of an operation (triggering condition of an operation for instance) or of an event (event predicate) or of an entity (entity constraint). If the real situation is that a copy can be loaned only if (a) "the copy is available and the subscriber is in good standing", the condition (a) should be described in the semantic network using a text type node.

4.3.2 Semantic meaning of edges

The aggregation, grouping and generalization abstraction forms are respectively represented by the "a", "r" and "g" edges. "a", "r" and "g" edges apply to domain, entity, operation and event nodes.

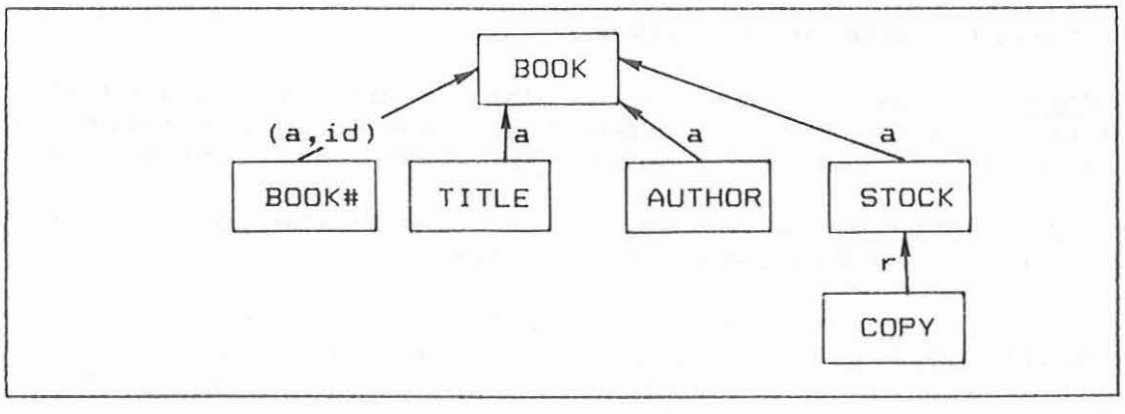
- An aggregation edge (edge a) may be defined either between two nodes of the same type or between two nodes of distinct types.

On example 1, subscribers are described as aggregate objects with 3 components: SUBSC#, NAME and ADDRESS.



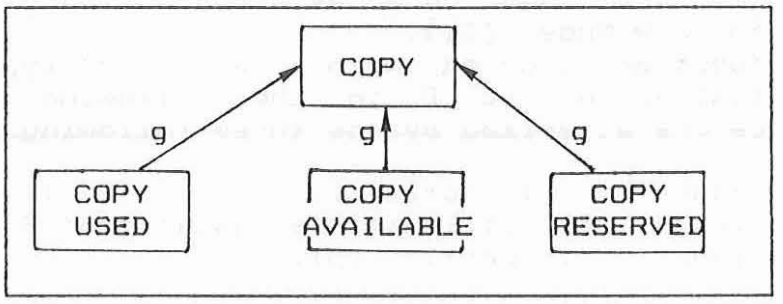
Aggregate components can be domain nodes or entity nodes. The "id" label precises the aggregate identifier (SUBSC# in the example).

- A grouping edge (edge r) is defined between two nodes of the same type. In the example below, a book is defined as a complex object; one of its components (STOCK) is a collection object. Every member of the collection is a copy. This means that a book in a library has usually several copies.



- A generalization edge (edge g) is used for the representation of a "is-a" relationship. An edge of type g is defined between two nodes of the same type. For example a copy can have three possible states; used, available and reserved described as 3 object nodes related to

the copy node by three "g" edges.



NOTE: In order to reach a good conceptual schema (criteria for "good" schema are presented in [30]), we have associated constraints (or norms) to nodes and edges of the semantic network which are part of their definition. We illustrate some of the norms with examples.

N1: Object identification

We consider as a good design discipline to identify each object of a conceptual schema. Thus, any entity node must have an in-going "id" labelled edge (see example 1).

N2: Operation atomicity

Operation atomicity (one operation is defined as acting on one and only one entity type) is required in the semantic network. Operation atomicity avoids redundancy in process description and thus inconsistency in process execution. Consequently an operation node is always origine of one "a" edge with an entity node as target node.

N3: Node / edge types compatibility norms

These norms avoid inconsistencies in the object constructions. The following figure represents the autorized edge types between two node types.

	DOMAIN	ENTITY	OPERATI.	EVENT	TEXT
DOMAIN	a r g	a			
ENTITY		a r g	a	a	
OPERATION			a r g	a	
EVENT				a r g	
TEXT	a	a	a	a	

Figure 4.3

N4: Cardinality norms

Cardinality norms are based on an extended notion of cardinality as introduced in the E/R Model [31].
Let f be a function coupled with a "a" or "r" type edge. For two entity nodes (called A and B in the following), the f and f-1 functions can be characterized by the three following properties.

- **totalness.** A function f is total (t) if and only if each instance of A is associated with at least one instance of B at any point of time, else the function is partial (p).

- **valuation.** A function f is single (s) if and only if each instance of A is associated with at most one instance of B at any point of time, else the function is multiple (m).

- **permanency.** The function is permanent (p) if and only if the set of instances of B associated to an instance a of A, at a time t is included in the set of instances of B associated to a at a time t' (t'>t), else the function is variable (v).

All combinations of properties are not allowed. The matrix on figure 4.4 summarized valid combinations of cardinalities. For instance the <tmp, tsp> couple of cardinalities for a "r" edge is allowed. Let us take an example, the function between the nodes STOCK and BOOK is total (t), multiple (m) and permanent (p), its opposite is total (t), simple (s) and permanent (p). This function can be associated with an "r" edge in the network.

	tsp	tsv	tmp	tmv	psp	psv	pmp	pmv
tsp	a		a		a		a	
tsv		a		a		a		a
tmp	r							
tmv								
psp								
psv								
pmp								
pmv								

Figure 4.4

4.4 The transformer base of rules

In order to help the designer to progressively improve the content of the semantic network during the design stage, the semantic network transformer uses essentially three classes of rules: diagnosis rules, improvement rules, mapping rules.

Let us concentrate and examplify the two first classes (mapping rules are used to map nodes and edges of the semantic network onto R-schema elements and are quite usual).

4.4.1 Diagnosis rules

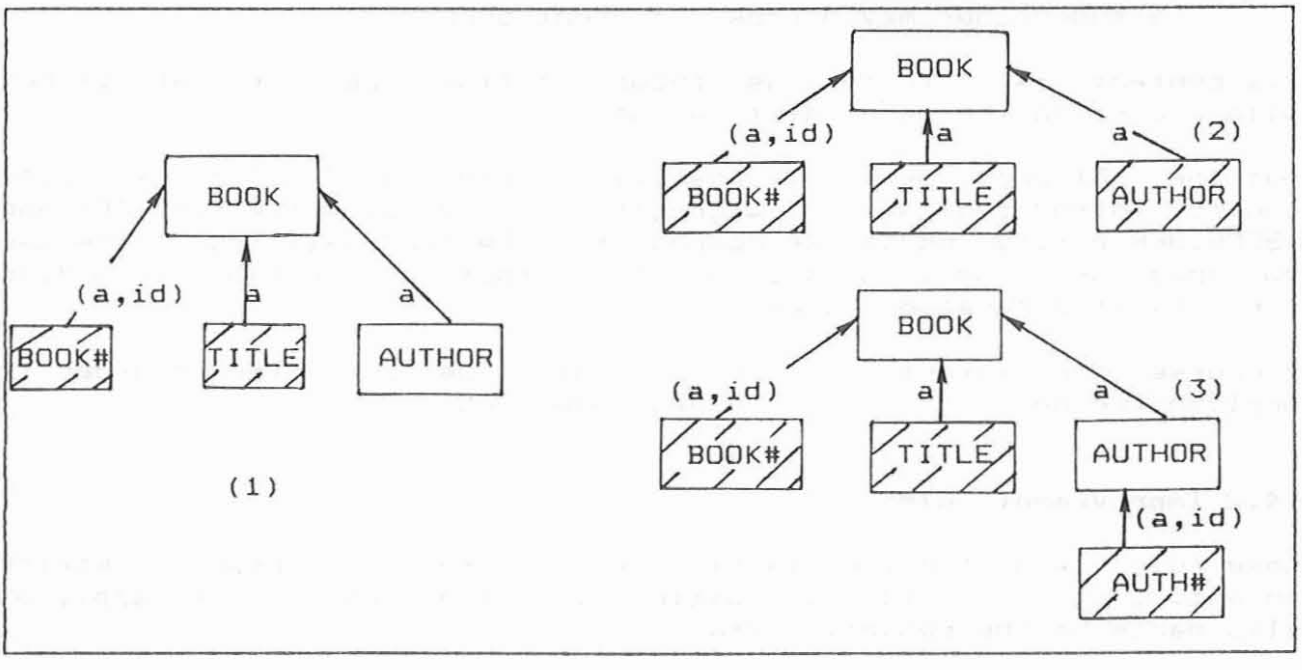
Diagnosis rules play a double role; they automatically detect errors in the semantic network and propose one (or several) solution(s) to correct each type of error.
Thus, using diagnosis rules, the tranformer is both a preventive and a creative tool.

Error detection is based on six design aspects, to which we have associated six groups of diagnosis rules.

- (a) object identification,
- (b) entity structuration,
- (c) event and operation structuration,
- (d) semantic network consistency,
- (e) semantic network completeness,
- (f) semantic network correctness.

Rules of type (a) and (b) are illustrated by examples.

Figure (1) describes part of a semantic network which will be detected as incorrect because of the non-identification of the AUTHOR entity node (the object identification norm N1 is violated).



Thus, the SNT will propose two alternative acceptable solutions:

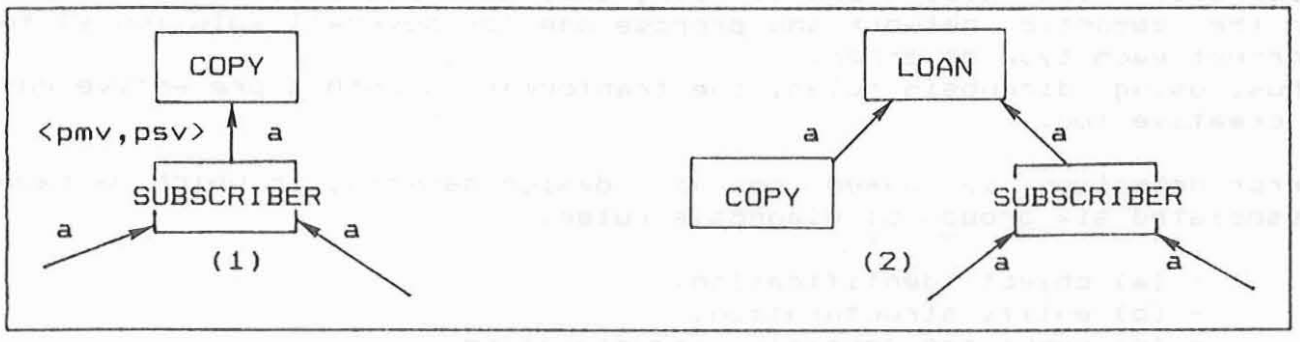
- Either to transform AUTHOR into a domain node (2),

- or to leave AUTHOR as an entity node, but necessarily identified by a domain node (AUTH#) (3).

The diagnosis / correction process is an interactive one. The SNT

will for instance in the previous case, explain to the designer (if required) that in the first solution authors will be considered only as property of books and not as independant entities.

In the following example the semantic network content illustrated in (1), means:



SUBSCRIBER is an aggregate entity node and a component entity node for the object COPY. The cardinalities of the relationship are as follows:

- <psv> a copy may be used by any subscriber,
  a copy may be used at most one subscriber,
  a copy may be used by distinct subscribers at distinct times.

- <pmv> a subscriber may borrow any book,
  a subscriber may borrow several books,
  a subscriber may borrow different books.

This content is detected as incorrect (the type of the arc is not valid according the cardinality norms).

Thus the SNT proposes an alternative description (2). The new entity LOAN is introduced as an aggregate entity with the two COPY and SUBSCRIBER entity nodes as components. The cardinalities of the two new edges are <pmp,tsp> and <pmp,tsp>; that is acceptable according to entity structuration rules.

Of course the interactive process will be activated in order to complete the description of the new node LOAN.

4.4.2 Improvement rules

These rules aim at giving facilities to improve the semantic network content. Contrarily to diagnostic rules, improvement rules apply on valid parts of the semantic network.

Basically these rules are formalization of design expert heuristics infered from the designer practical experience. They are based on pattern recognition and suggest for each initial pattern of the semantic network a more sophisticated one (or several alternative ones) according to some specific design discipline.

For instance as we will illustrate later the SNT can try to sophisticate the representation of entity classes using

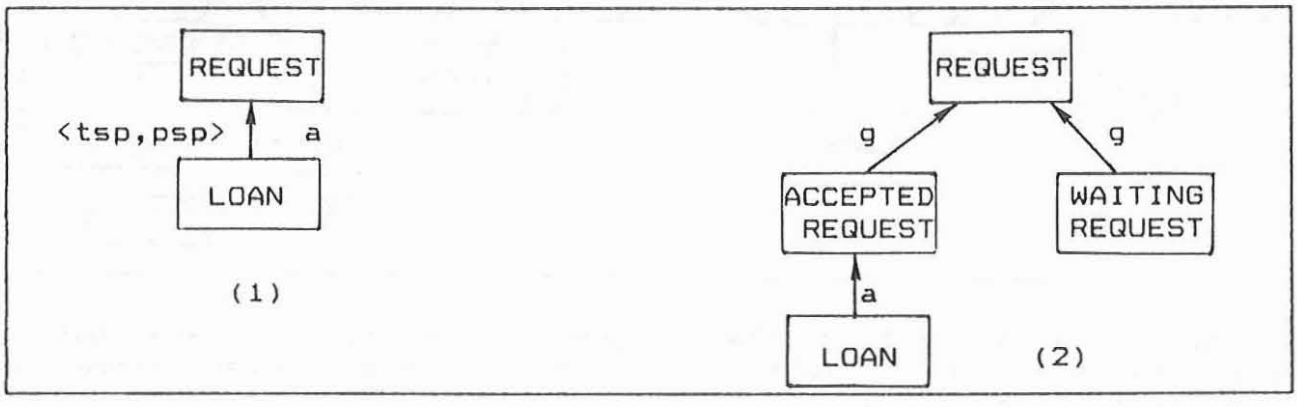
specialization; or it can suggest a more complete representation of entities based on temporal reasoning; or even combine these two aspects.

Improvement rules relate to:

- historization of entities and relationships,
- specialization (of entity, event and operation types),
- behaviour completion,
- domain structuration.

Similarly to diagnosis rules, improvement rules identify a specific pattern in the semantic network and propose to the designer one or several improved representations pointing out some, may be; forbidden or undertaken design problem. Let us give two examples of semantic network transformations.

Situation (1) corresponds to a pattern identified with a <tsp, psp> couple of cardinalities.

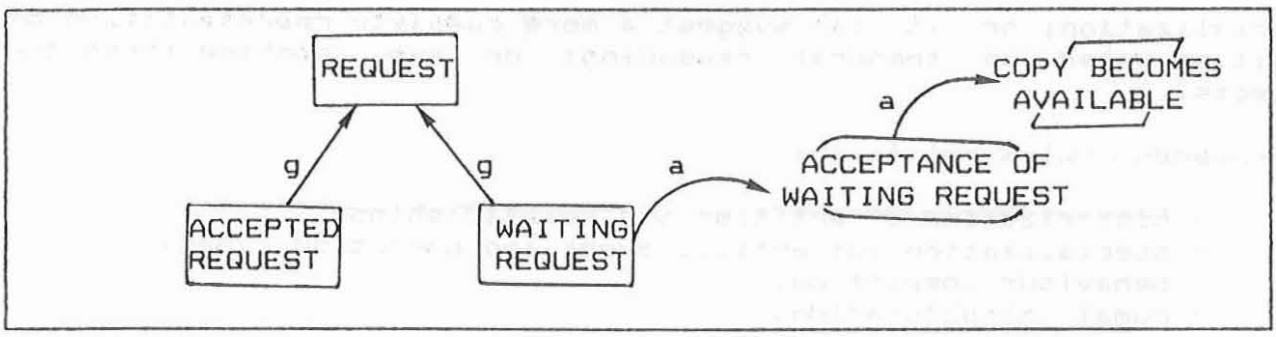


In this example, two entity nodes REQUEST and LOAN are aggregated in such a way that:

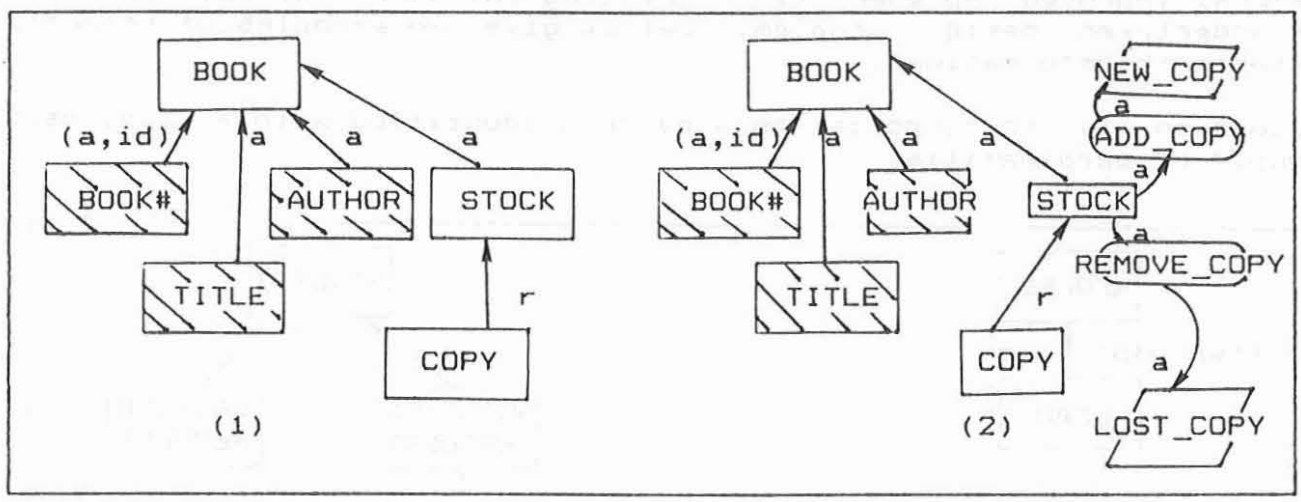
- a loan is associated with one and only one request; furthermore a loan is always associated with the same request (tsp),

- a request may be "not accepted"; thus it is not associated to a loan (psp).

The SNT proposes to improve the description presented in (1). This solution suggests to distinguish "accepted requests" and "waiting requests" that have, probably, different and specific operations and events. Following this line the designer can complete the new network. For instance he can add the operation node ACCEPTANCE WAITING REQUEST. This new operation node is defined on REQUEST WAITING. In this context, the designer must find the event type that triggers this operation type. In the example, the event type is COPY BECOMES AVAILABLE. Finally the designer reaches a more complete and precise description of reality summarized as follows:



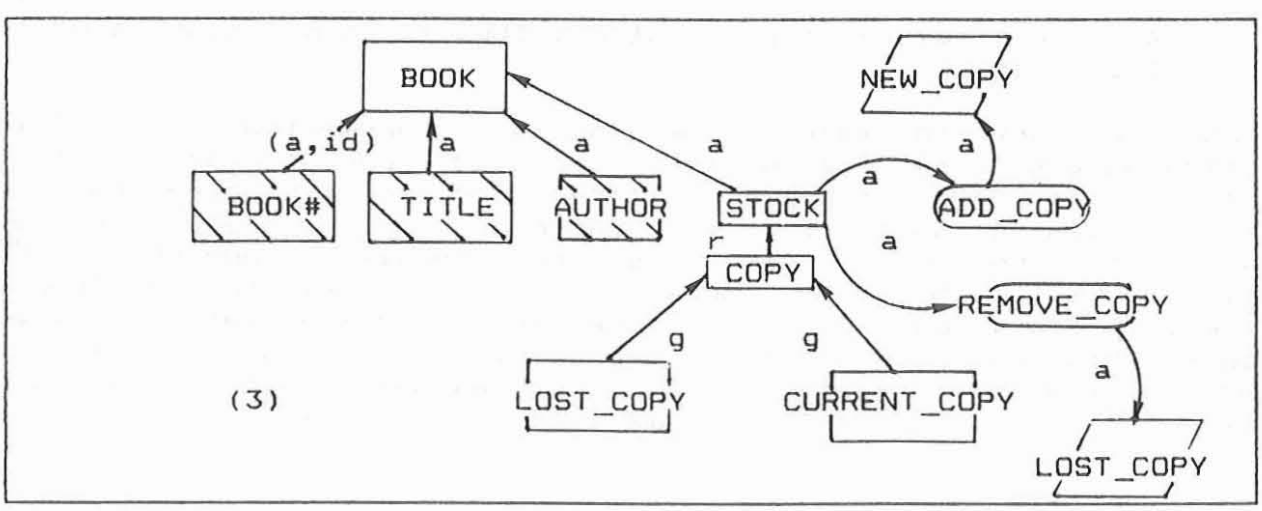
The following example relates to behaviour completion and historization of entities. Let us consider situation (1):



(a) This rule suggests to the designer to describe the behaviour of the entity type STOCK. STOCK is a collection object. Every member of the collection is a copy.

The representation (2) proposes two modification events on STOCK: LOST_COPY and NEW_COPY.

(b) On this new representation, the SNT can apply a new improvement rule which suggests to the designer to memorize the lost copies. The new proposed representation (3) uses the generalization/ specialization structure.



During the design process, the EDT drives actively and intelligently the designer, and progressively improve the semantic network until reaching a satisfactory solution for the designer and for the EDT. Then, the semantic network can be mapped onto R-schema elements, which can be used by the prototyping tools.

5 PROTOTYPING TOOLS

Information System prototyping is based on an automatic management of the database dynamics specified in the R-schema.
This involves :

- automatic recognition of events;
- automatic triggering of appropriate operations when an event occurs;
- operation execution control;
- event synchronization.

To attain such an automation, we have chosen to :

a) use a relational DBMS to deal with :
   - managing the relations of the meta-base corresponding to the R-schema specification;
   - executing operations texts and evaluating factors, conditions and predicates : this requires an interpreter more powerful than a simple SQL interpreter.

b) develop a mechanism able to :
   - recognize an event;
   - determine which operations to execute;
   - trigger and control operations execution;
   - synchronize event-chaining.

This mechanism is similar to the inference engine of a forward chaining expert system, whose cyclic function is to :

- test the rule premisses;
- choose a candidate rule;
- execute the action-part of the rule;

and which possesses a rule-chaining strategy.

The mechanism we propose is composed of four units managing all kinds of events.

- the **temporal processor** recognizes temporal events;
- the **event processor** recognizes internal events and processes all events and their synchronization;
- the **PROQUEL interpreter** executes all texts of predicates, conditions, factors and operations when required by the event processor;
- the **application monitor** allows to introduce instances of external events as test cases for prototyping.

We focus now on the event processor which is the key part of the prototyping tool.
The event processor fulfils three main functions :

- takes into account external and temporal events;
- processes events;
- orders them.

The first function is based on a FIFO management of the Message Queue. The second function consists of a meta-base search for appropriate conditions, factors and operations that will be evaluated or executed by the relational DBMS. These two functions do not present any major difficulties, as opposed to the third function, presented in the following section.

The chosen strategy for event synchronization is based on the **induction** notion, and on the use of the **induction graph**, which is derived from the R-schema.

5.1 The induction notion

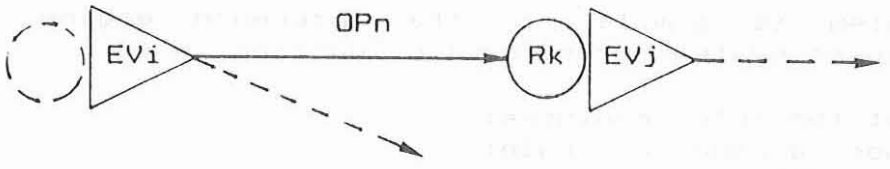
The induction notion is used to point out the ordering of events from the R-schema.

DEFINITION :

An event EVi **inducts** an event EVj, if and only if :

- EVi triggers OPn which modifies the relation recognized by EVj,
- an occurrence of EVi, followed by the execution of OPn can produce an occurrence of EVj.

Graphically, the situation is the following :



The notation used to represent an induction is : EVi $\xrightarrow{OPn}$ EVj

5.2 The Induction Graph construction

The Induction Graph uses the above notation. It contains :

- nodes representing R-schema events,
- directed edges representing inductions,
- weights on the edges, which represent operations and are used as "induction conditions".

The Induction Graph construction is accomplished in two steps :

- an automatic step, producing the Maximal Induction Graph,
- a manual step transforming the Maximal Induction Graph into the Induction Graph.

1st STEP

The Maximal Induction Graph can be automatically deduced from the R-schema :

- "a priori possible chainings" are obtained by analysing the **ON**, TYPE and **TRIGGER** parts of event and operation specifications. For a given event EVi, the chain is composed of all those events ascertaining relations modified by the operations triggered by EVi,

- in order to keep only "structurally possible chainings", the occurrence of each operation's **TYPE** (INSERT, DELETE, UPDATE) is checked within the **ON** part (i.e the category) of the internal event(s) it seems to induce. So, impossible chainings like "a product deletion produces a new availability" will be removed from the graph.

Figure 5.1 presents the induction graph corresponding to our case study.

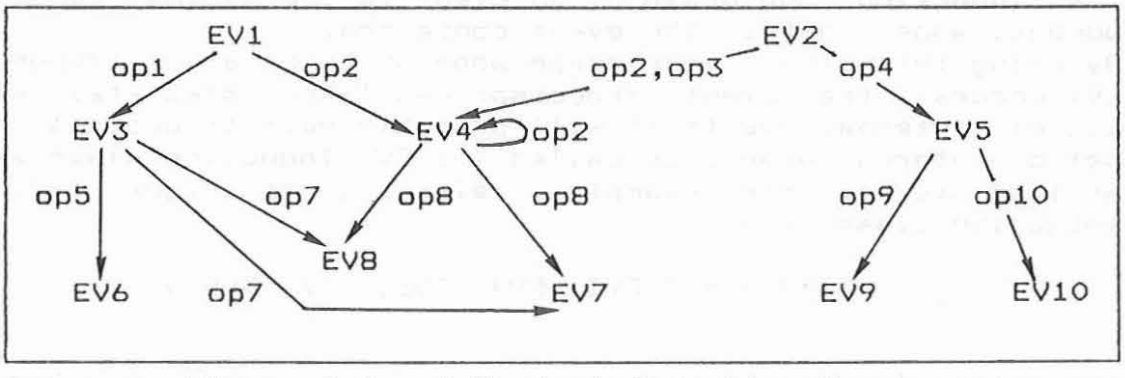


Figure 5.1 : A Maximal Induction Graph

2nd STEP

The designer then manually modifies the Maximal Induction Graph, until he obtains the final Induction Graph.
During this step, the designer removes all the chainings that seem impossible to him from the graph. For example, EV4 ("loan request") seems to induce EV8 ("copy availability"). This is because EV4 triggers a modify operation on COPY (op8) and EV8 recognizes insertions or modifications of the COPY status.
In reality, an EV4 occurrence will never generate an EV8 occurrence since op8 always put the COPY status to "LOANED", and EV8 only recognizes modifications setting a COPY status to "AVAILABLE".
This kind of "false induction" cannot be detected automatically since it involves a semantical interpretation of predicates, conditions, factors and operations.

The final Induction Graph is an optimized and generally non-connected graph, which contains only "semantically possible chainings". Figure 5.2 presents the Induction Graph corresponding to the Maximal Induction Graph of figure 5.1.
If there are cycles in the Induction Graph, they are detected automatically, and the designer is asked to a confirm an "impossible endless loop".

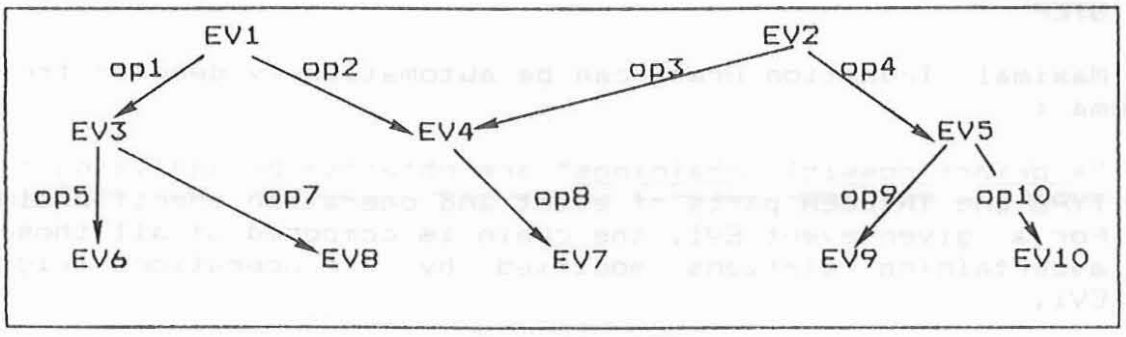


Figure 5.2 : Induction Graph

5.3 Internal event chaining strategy

Given an external or temporal event to be processed, the chosen strategy is based on a "breadth-first" traversal of the event Induction sub-graph.

** The **induction sub-graph** of an event is the maximal connected component, whose root is the event concerned.

By using this kind of sub-graph when an external or temporal event EVi occurs, the Event Processor can learn immediately what "the set of internal events it will probably have to process" is. This set of internal events is called the EVi **Induction Class** and is written C(EVi). For example, referring to figure 5.2, the EV1 Induction Class is :

C(EV1) = { EV3, EV4, EV6, EV7, EV8 }

** The **internal event sequence** construction is based on a breadth-first traversal of the Induction sub-graph.

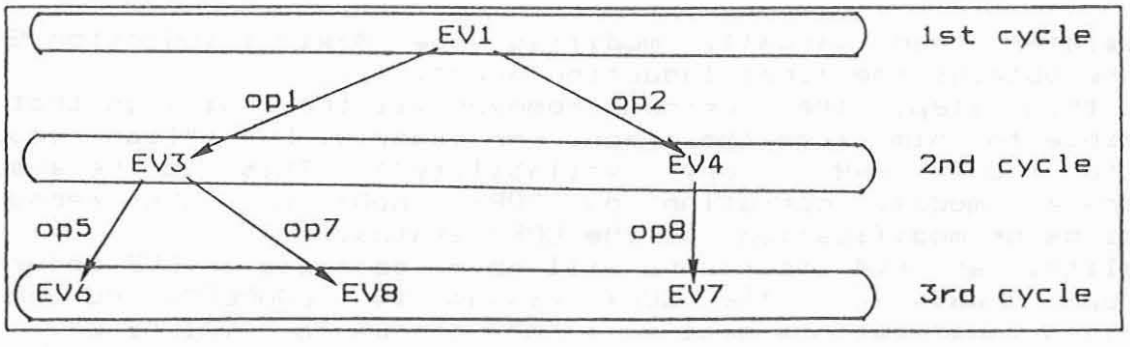


Figure 5.3 : Internal event sequence

For example, if EV1 occurs, the complete processing cycle will include

1st cycle: EV1
2nd cycle: EV3 + EV4
3rd cycle: EV6 + EV7 + EV8

It means that within each cycle, all events from the same level are processed.

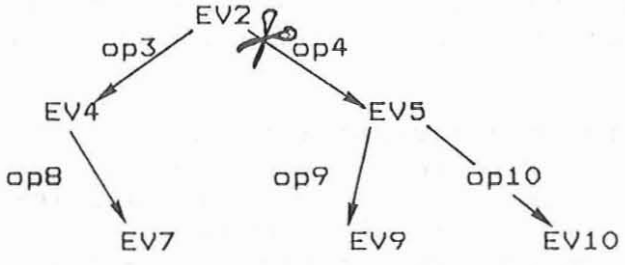
The "breadth-first" strategy (e.g EV1, EV3+EV4, EV6+EV7+EV8) has a real advantage over a "depth-first" (EV1, EV3, EV6, EV4, EV7) or a "random" strategy (EV1, EV3, EV8, EV4, EV7, EV6). In fact, this strategy permits optimal management of the input/output implicit parameters. Idle time between :

- generation of an "operation output parameter",
- and its use as input parameter to process the event induced by this operation, is minimal.

Internal events are recognized as soon as "noticeable state changes" occur (in fact : just after all operations triggered at the same level have been executed); and these events are processed as soon as they are recognized (i.e during the next basic cycle of the event processor).

In this manner, there is **no parameter waiting for use** during a complete basic cycle. This is not true with other strategies; for instance, in the "depth-first" strategy, the EV4 input parameters must be kept in memory as long as EV3 and EV6 are still being processed.

The purpose of the Induction Graph is an optimization of the Event Processor work. For example, refering to figure 5.2, during the processing of ev2, if OP4 isn't in the list of operations to execute, a whole part of the EV2 Induction sub-graph can be pruned off :



So it permits :

- avoidance of useless predicate tests (EV5, EV9, EV10),
- avoidance of useless parameter recording,
- an earlier freeing of resources ("read-locked" relations for predicate, condition and factor evaluation; "write-locked" relations for operation execution).

It appears that an external or temporal Induction Class will become smaller and smaller after each basic cycle, and will finally reach the empty state. (another external or temporal event will then be processed.)

At any moment, the Event Processor knows what it is processing and what it must deal with next; so it controls the whole process fully.

USING THE PROTOTYPE

During the experimentation of the prototype, the Event Processor displays a trace of what it is doing : which events are currently processed or waiting, which conditions are true, which operations

have to be executed, and so on.

Analysing this trace, the designer can easily detect if things are going wrong (bad result for a condition, database error on a predicate evaluation, occurrence of a wrong event, ...) and immediately correct the errors, modifying the specifications through one of the design interfaces.

Applying an iterative strategy, the designer will progressively refine his specifications until the behaviour of the prototype is fully correct for him and for the end-users.

6. CONCLUSION

In this paper, we have presented an integrated computer aided tool so-called RUBIS, for designing and prototyping Information Systems.

The RUBIS-schema is a declarative specification of the IS, made using the formal specification language called PROQUEL (PROgramming QUEry Language). The R-schema provides a conceptual description of both the static and dynamic aspects of the IS to be built. The static IS aspects are modelled by **relations** while **operations** (elementary actions on an object) and **events** (elementary state changes triggering one or several operations) allow the modelling of the dynamic aspects of objects. Thus, the R-schema is a collection of relations, operations and events specifications. In this schema the temporal aspects of the application are also taken into account; they are modelled by using the time types and functions provided by the RUBIS Time Model.

RUBIS includes **three different interfaces** as computer aided design tools: a Graphic Interface, an Expert Design Tool working on a "complex object" description of the application domain and a Menu Interface based on the PROQUEL language.
The Graphic and Menu based interfaces are both devoted to experienced designers while the Expert Design Interface can be used by relatively unexperienced analysts or designers.

Prototyping in RUBIS is achieved by the Temporal Processor which manages temporal aspects of the specification and by the Event Processor which manages event recognition and synchronization. Both Processors use the PROQUEL interpreter.

A first version of RUBIS is running on SUN Workstation, under the UNIX system. Prototyping tools are written using the C language, and design tools are using the X-Windows system. The expert design interface is partly written in Prolog.

Current developments are leading towards :

- the development of a functional debugger for the prototyping aspects;
- the extension of PROQUEL to a real persistent language;
- the provision for the implementation of code generators which will translate the PROQUEL specifications into a target "embedded language" (Pascal/SQL, C/QUEL, ...);

- the development of a graphical interface to manipulate the Semantic Network;
- the extention of the expert design interface to include tutorial functionnalities.

REFERENCES

[1] C. ROLLAND, C. RICHARD: The Remora methodology for information systems design and management in IFIP WG8.1 working conference on "Information systems design methodologies: a comparative review" 1982

[2] BOUCHET and al. : "Databases for Microcomputers : the PEPIN Approach" ACM SIGMOD/SIGSMALLS, Orlando, Florida, Oct.1981.

[3] ROLLAND C, BENCI G, FOUCAUT O : "Conception de Systèmes d'Information : La Méthode REMORA", Eyrolles 1987.

[4] BUBENKO J.A. : "The temporal dimension in Information Processing" in Architecture and Models in Database Management, G.M. NIJSSEN, ed. North-Holland (1977).

[5] WIEDERHOLD G., FRIES J.F., WEYL S. : "Structured organization of Clinical Databases" Proc. of AFIPS National Computer Conf., Anheim, 1975.

[6] BOLOUR A., ANDERSON T.L., DEKEYSER L.J. and WONG H.K.T. : "The role of time in information processing : A survey" ACM SIGMOD RECORD vol. 12, n° 3, April 1982.

[7] SNODGRASS R. : "The temporal query language TQUEL" ACM Transactions On Databases Systems, vol. 12, n° 2, June 1987.

[8] NAVATHE S.B., AHMED R. : "TSQL : A language interface for history databases" AFCET-IFIP WG8.1 TAIS Conf., Sophia-Antipolis, France, May 1987.

[9] CODD E.F. : "A Relational Model of Data for Large Shared Data Banks" Communications of the ACM, vol. 13, n° 6, 1970.

[10] M.R. GUSTAFSSON, J.A. BUBENKO T. KARLSSON: "A Declarative Approach to Conceptual Information Processing" in IFIP WG8.1 Working Conference on "Information Systems Design Methodologies: a comparative review" 1982.

[11] ANDERSON T.L. : "Modeling time at the conceptual level" Proc. 2nd International Conf. on Databases, Jerusalem, June 1982.

[12] SNODGRASS R., AHN I. : "A taxonomy of time in databases" Proc. of ACM SIGMOD 85, Mar. 1985.

[13] ZLOOF M.M. : "Query By Example : a database language" IBM Systems Journal, vol. 16, n° 4, 1977.

[14] OVERMYER R., STONEBRAKER M. : "Implementation of a time-expert in a Database System" ACM SIGMOD RECORDS, vol. 12, n° 3, Apr. 1982.

[15] ADIBA M., BUI QUANG N., PALAZZO J. : "Time concepts for generalized data bases" ACM Annual Conference, Denver, Colorado, U.S.A., Oct. 1985.

[16] AHN I., SNODGRASS R. : "Performance evaluation of a Temporal Database Management system" Proc. of ACM SIGMOD Conf., 1986.

[17] DADAM P., LUM V., WERNER H.D. : "Integration of Time Versions into a Relational Database System" Proc. of 10th VLDB, Singapour, Aug. 1984.

[18] ADIBA M., BUI QUANG N. : "Historical multi-media databases" Proc. of 12th VLDB, Kyoto, Japan, Aug. 1986.

[19] BARBIC F., PERNICI B. : "Time modeling in Office Information Systems" Proc. of ACM SIGMOD Conf., Austin, Texas, May 1985.

[20] CLIFFORD J., RAO A. : "A simple general structure for temporal

domains" AFCET-IFIP WG8.1 TAIS Conf., Sophia-Antipolis, France, May 1987.
[21] BOLOUR A., DEKEYSER L.J. : "Abstractions in temporal information" Information Systems, vol. 8, n° 1, 1983
[22] DBE : Data Base Engineering review vol 17, n°4, Special issue on data design aids methods and environment, Dec. 84
[23] S. CERI: "Methodologies and Tools for Database Design" ed, North Holland Publ Co 1983.
[24] R.P. BRAGGER, A. DUDLER, J. REBSAMEN, C.A. ZEHNDER: "GAMBIT: An interactive Database Design Tool for Data Structures, Integrity Constraints and Transactions" in Database Techniques for Professional Workstations, ETH Zurich 1983.
[25] J.M SMITH, C.P. SMITH: Database Abstractions : Aggregation. Communications of ACM. June 1977.
[26] J.M. SMITH, C.P. SMITH: Database Abstractions : Aggregation and Generalization. ACM TRANSACTIONS on Database Systems. June 1977.
[27] M.L BRODIE, E. SILVA : Active and Passive Component Modeling: ACM/PCM in IFIP WG8.1 working conference on "Information systems design methodologies: a comparative review" 1982.
[28] KAHN K., GORRY G. A. : "Mechanizing Temporal Knowledge" Artificial Intelligence, Vol.9, N°1, Aug. 1977.
[29] MITTAL S. : "Event-based Organization of temporal Databases" Proc. CSCSI/SCEIO Conf. 82, Saskatoon, Saskatchewan, 17-19 May 1982.
[30] CAUVET C. : "Un modèle et un outil d'aide à la conception des Systèmes d'Information" Ph. D. Univ. of Paris 6, Nov. 1988.
[31] CHEN P. : "The Entity/Relationship model : towards a unified view of data", ACM TODS, Vol 1, n° 1, 1976.
[32] NOBECOURT P., ROLLAND C., LINGAT J.Y. : "Temporal management in an extended relational system" 6th British National Conf. on Databases, Cardiff, G.B., July 1988.
[33] LINGAT J.Y, NOBECOURT P., ROLLAND C. : "Behaviour management in database applications" VLDB 1987, Brighton, G.B., Sept. 1987.
[34] LINGAT J.Y, COLIGNON P, ROLLAND C. : "Rapid prototyping : the PROQUEL language" VLDB 88, Los Angeles, USA, Sept 88.