CORRECTION OF CONCEPTUAL SCHEMAS

C. SOUVEYET, C. ROLLAND***

* Laboratoire MASI, Universite Pierre et Marie Curie
4 place JUSSIEU, 75235 PARIS, FRANCE
Fax : + 33-1-46-34-19-27

** Universite de la Sorbonne
17 rue de la Sorbonne, 75231 PARIS cedex 05, FRANCE

Abstract :

This paper presents the interim results of a research project aimed at the prototyping of an automatic tool, *Rubis*, to aid in the development of, validate and correct the conceptual specification of information systems.

The Rubis systems allows a designer to specify an information system using the Proquel language and to subsequently execute the specification in order to prototype the design.

We present the control rules which enable the diagnosis of the final specification, called an *R-Schema*, and describe the help available to assist the designer in correcting mistakes and anomalies detected during the diagnosis.

I Introduction

In this paper we introduce an environment called Rubis [ROL88] [LIN88b], which aims to aid the designer in designing information systems. The Rubis system provides the designer with:-

a model and associated high level development language, Proquel, to aid in the development of the specification of the R-Schema, specifying the static, dynamic and temporal aspects of the information system,

functions to determine the correctness of the R-Schema by running various checking rules on it, called the *Validation Module*

functions to assist in the correction of any errors highlighted by the checking phase, called the *Correcting Aid Module*

a prototyping mechanism allowing the execution of the specification on test cases, thus validating the dynamic aspects of the application. This is seen as an aid in improving the dynamic aspect of the final application,

various interfaces to modules implementing the above functions.

The paper is structured as follows:-

Part II discusses the Rubis architecture and functionality

Part III discusses the checking rule architecture, the checking rule taxonomy and the control levels used in checking the correctness of the R-Schema

Part IV discusses the Correcting Aid Module.

We conclude with a conclusion in Part V.

II Rubis Architecture and Functionality

As shown in figure 1, the Rubis system has four components as follows:-

1. The R-Schema: it describes the information system, and is stored in the Meta-base.

2. The R-Schema design interfaces: the Menu Interface, the Graphical Interface and the Proquel Interpreter.

3. The prototyping tools: the Application Monitor, the Event Processor, the Temporal Processor, and the Proquel Interpreter.

4. The validating tools:the Validation Module and the Correcting Aid Module.

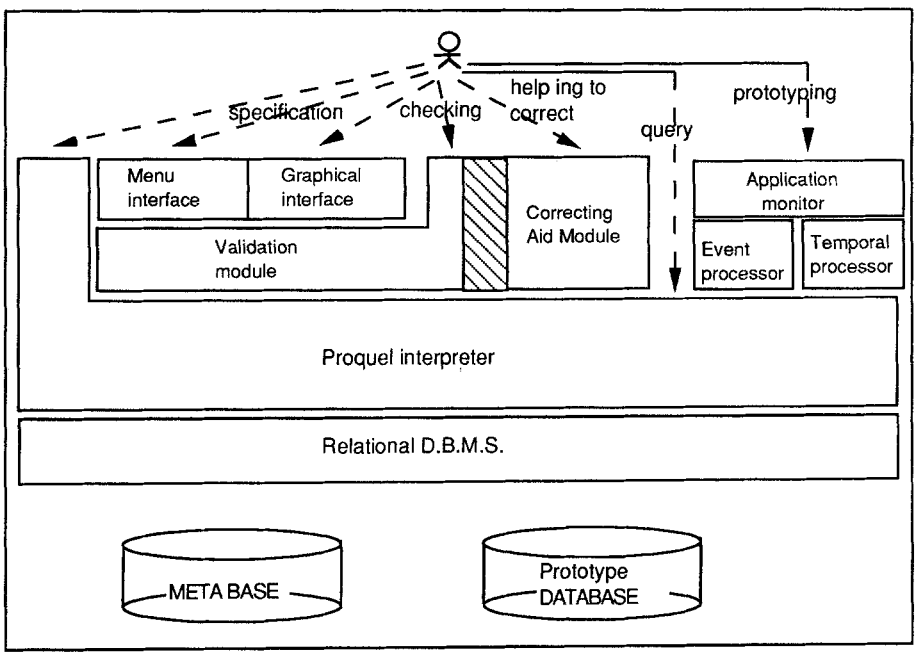

Figure 1 : Architecture of the Rubis system

II.1 The R-Schema

The R-Schema is based on the model used in the Remora methodology [ROLL.82] [ROLL.87], and describes both static aspects (structure) and dynamic aspects (behaviour) of the application. It is stored in relational form [COD.70] in the Meta_base, and is the focal point of interaction between the designer and the Rubis system.
The static aspects are modeled using objects, representing entities or entity associations in the real world (e.g. client, invoice, loan, etc.), and integrity constraints associated with these objects.

The constraints are classified in different classes; referential constraints, cardinality constraints, and domain constraints.

The dynamic aspects are modeled using :

- operations which represent elementary actions on an object (e.g. add a new client, modify an order, etc.),
- events which represent elementary state changes in the system at which time some operations must be triggered (e.g. when an order arrives, insert the order into the database, reserve the requested goods, prepare the delivery, etc.). The state change description of an object is defined in the event predicate.

A distinction is made between external events, which model the arrival of a message from the real world, internal events, which model elementary state changes of an object, and temporal events, which represent temporal conditions under which certain processing is triggered.

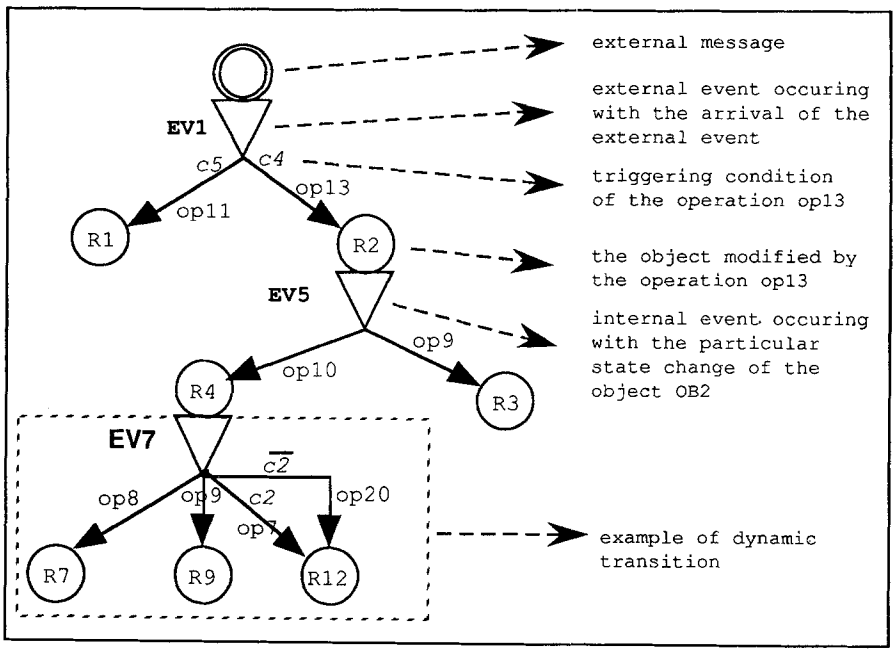

Figure 2 : a representation of a dynamic graph

The Temporal aspects of the application are modeled using the temporal functions and types of the Rubis Temporal Model [NOB.88].
The R-Schema is therefore a collection of relations, events, and

operations defined for an application using Proquel specifications. The content of the R-Schema can be illustrated using a graph (Fig 2). Such a representation introduces the *dynamic transitions* of the application, showing their sequences and precedences. A dynamic transition is composed of (1) an event, (2) all operations triggered by the event, and (3) all references to objects modified by these operations. This corresponds to an elementary database transaction, since by definition a Rubis dynamic transition is atomic and must maintain database coherency across database coherency across database changes.

II.2 Design Tools

The Menu Interface allows the insertion, modification and deletion of different components of the R-Schema. Components are manipulated by the designer filling in forms during the specification process.

The Graphical Interface gives a great freedom to the designer during the acquisition stage of the specification of the R-Schema. It integrates a Graphical Editor which facilitates the drawing of the static and dynamic schemas.

The Proquel Interpreter is a design tool and a prototyping tool. Proquel [LIN.88a] is a specification language, a data manipulation language and a programming language. As a design tool, the Proquel Interpreter provides statements to insert, modify, and delete components of the R-Schema. The next section describes the Proquel interpreter as a prototyping tool.

II.3 Prototyping Tools

The Application Monitor allows the definition of the end-user interface. It automates the generation of data input screens, corresponding to each external event defined by the designer, from the specification text of these events. This text serves to specify the structure of the received message, and hence it may be used as a specification of the end-user screen. The associated event generated screens allow for the inputting of data test cases to test the correctness of the R-Schema behaviour.

The Temporal Processor manages all temporal aspects of the application, including :

-handling attributes of type 'TIME' (timestamps, dates, chronological order, calendar conversion, etc..),
-historical processing,
-evaluating expressions using temporal functions and types,
-automatic recognition of temporal events (absolute dates, periodic events, events times relative to other events, etc..).

The Event Processor drives the prototype. It facilitates the execution of the R-Schema by sequencing and synchronizing dynamic transitions, including :

-handling instances of external and temporal events,
-evaluating the triggering conditions of operations,
-controlling the execution of operations when the triggering condition is satisfied,
-recognition of internal event instances,
-managing the transaction aspects of the application.

The Proquel Interpreter is viewed here, as a prototyping tool. It is used by each of the other modules, but in particular by the event processor for the execution of operation, condition, and event predicate texts.

The relational DBMS [BOUF.86] is the foundation of the Rubis system. It manages the relations in the prototype database as well as the relations in the Meta-base containing the R-Schema.

II.4 The Validation Tools

The Validation Module performs the validation of the R-Schema, detecting situations which are either incorrect or probably incorrect.

The Correcting Aid Module aids the designer to correct the anomalies detected by the Validation Module.

R-Schema diagnosis requires a set of checking rules, which are presented in the next section. We, then describe two other aspects, the strategy used in anomaly detection, and the help provided to the designer in correcting these anomalies.

III The Checking Rules

The Validation Module is based on the set of checking rules to contr the correctness of the R-Schema.

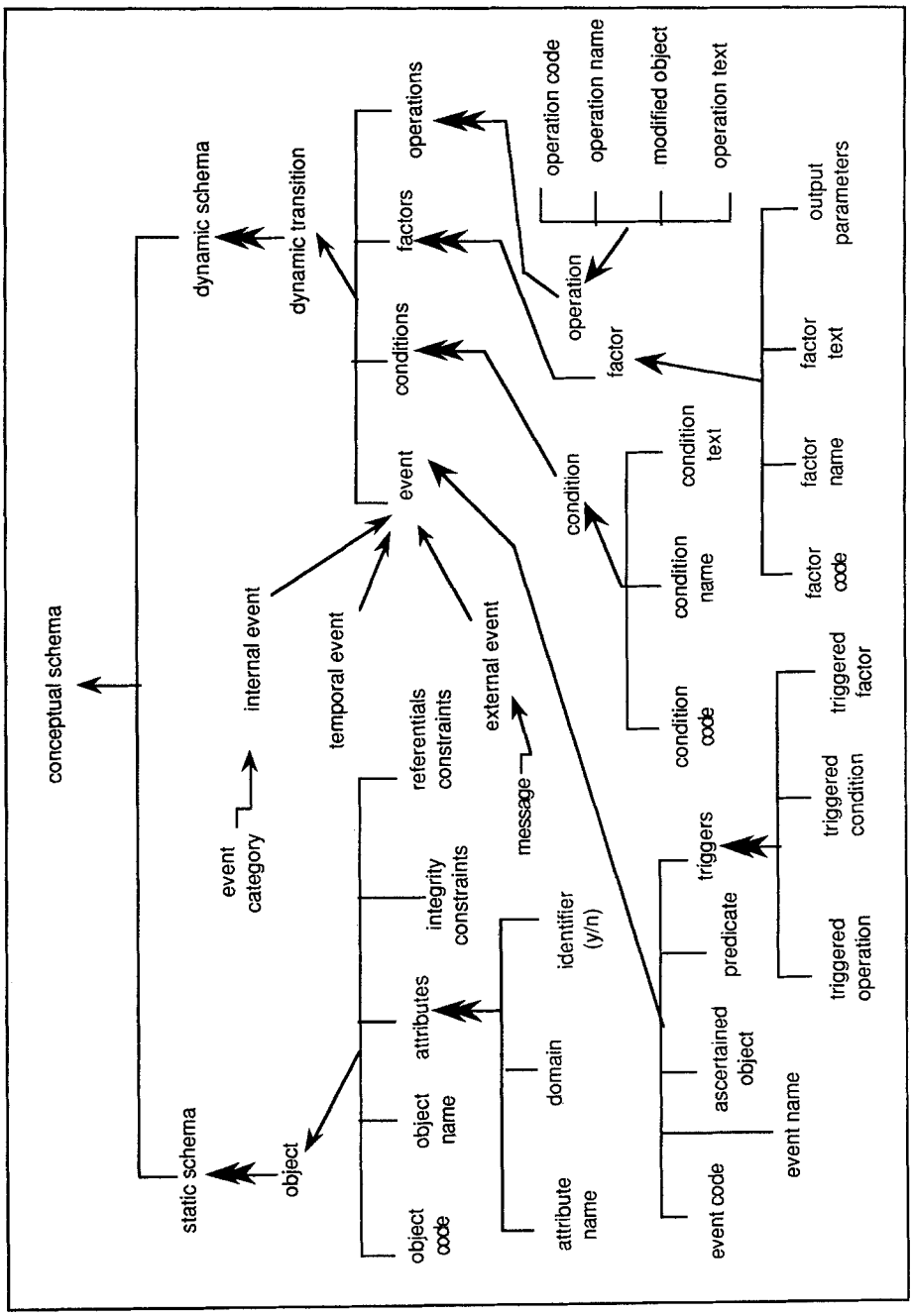

Figure 3 : Meta-Base design

The checking rule can be viewed as a predicate which must be satisfied by the R-Schema. In this section, we present the taxonomy of these rules, the implementation of them and the control levels at which they belong.

III.1 The Checking Rules Architecture

The aim of this architecture is to provide the Validation Module with the sequence of control rules to be checked at the requested level explained below (cf section III.4).

The strategy, we have chosen, is to model the static aspects of all components to be controlled and to associate to each component the set of checking rules which validate it. We have used on one hand, the aggregation, generalization and association constructors of semantic models [BRO.82] [BRO.83] [CAU.88] to model the static aspects of the R-Schema, whilst on the other hand, we have used the "encapsulation" notion defined in the object-oriented approach [PIN.88], to associate the checking rules to the component which they validate. The figure 3 illustrates the principal static part of the Meta-base which is checked.

For each constructor (aggregation, generalization and association), we define a checking rule strategy. Consequently, this hierarchical organization of components implicitly defines the control execution order. We illustrate this mechanism by an example of an "aggregate" component.
Consider the sub-set illustrated by thè following figure :

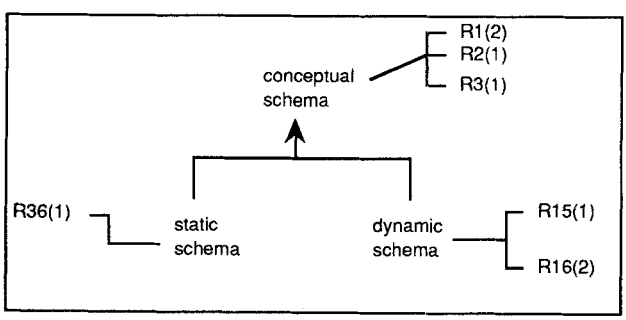

Figure 4 : representation of an aggregate component

This defines the component "conceptual schema" as an aggregate of two components "static schema" and "dynamic schema".

On the "static schema", we take the control rules which validate the static part without reference to the dynamic part. For instance, rule R36 is a method attached to "static schema" which expresses that **"when two objects have the same identifier, we must integrate both objects attributes in the only one object"**.

On the "dynamic schema", we take the control rules which validate the dynamic part without reference to the static aspects. For instance, the rules R15 and R16 express respectively that **"Each dynamic transition of an internal event must depend chronologically on an dynamic transition of an external or temporal event"** and **"When two dynamic transitions of internal events ascertain state changes of the same object, their predicates must be exclusive"**.

On the "conceptual schema", we take the rules which validate the relationship (or reference) between "static schema" and "dynamic schema". For instance, the rules R1, R2 and R3 are attached to "conceptual schema". The expression of these rules are :

R1 : **"Each object in the "static schema" must also exist in the "dynamic schema"**.

R2 : **"Each object on which an event is ascertained, must be defined as an object in the static schema"**.

R3 : **"Each object modified by an operation must be defined as an object in the static schema"**.

The execution order of controls for "conceptual schema" is simply deduced from the component structure. We execute the control attached to the "static schema" and the controls attached to the "dynamic schema" in any order and when the components are correct, we execute the control of "conceptual schema" to validate the cross-references between "static schema" and "dynamic schema". We illustrate this mechanism by the following example.

If we define the following R-Schema :

-The objects OB1, OB2 and OB3 are described, but OB1 and OB2 have the same identifying attribute,

-Two events EV1 and EV2 are defined. EV1 is an external event which triggers the operation OP1 and EV2 is an internal event which ascertains the state change of the object OB2 and triggering the operation OP2.

-Two operations OP1 and OP2 are defined and they modify respectively OB1 and OB4.

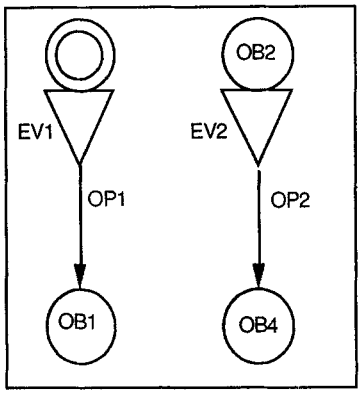

Figure 5 : The dynamic part of the R-Schema

The validation of the "conceptual schema" at level 2 will occur as follows:
We apply the rules attached to the components of the conceptual schema". Rule R36 is not respected on the "static schema" because OB1 and OB2 have the same identifying attribute and the rule R15 is not respected because the internal event EV2 does not depend on an external or temporal event. So, the "conceptual schema" is incorrect. If we correct this schema as follows :

-the event EV2 ascertains the state change of the object OB1, then the R-Schema satisfies rule R15,

-the changing of the identifying attribute of the object OB2 implies that it becomes different from the identifying attribute of OB1, then the R-Schema satisfies rule R36.

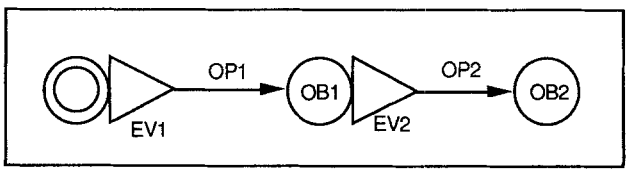

Figure 6: Graphical representation of the corrections

When we re-run the checking of the R-Schema, an error is detected by rule R3 because the operation OP2 modifies the object OB4 which is not defined as an object in the static part. If we correct the definition of the operation OP2 for modifying the object OB3, the "conceptual schema" becomes correct.

A similar approach, is applied for the "set" component and the "generic" component. In the case of the "set" component, illustrated by the figure 7, the checking module translates the "set" structure by on the one hand, the iterative control function which validate each set members (for instance, we check each "dynamic transition"), and on the other hand, the execution of controls attached to the "set" component (for instance, we check the correctness of "dynamic schema").

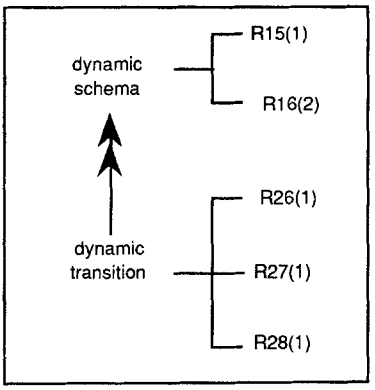

Figure 7 : Schema of a set component

The case of a "generic" structure, illustrated by figure 8, is more complex.

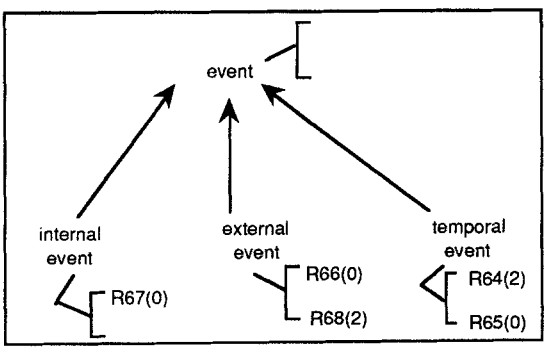

Figure 8 : Schema of a generic component

We attache to the "specialized" component the rules which allow the validation of :

-its own components,

-the relationships between them,

-the relationships between the components of the "generic" item and

the components of the "specialized" item which corresponds to it.

This structure is translated by a checking function which first runs the controls attached to the "generic" component (e.g. we check the component "event") and then the controls attached to the "specialized" component which corresponds to it (e.g. we check the "specialized" component "internal event" or "temporal event" or "external event").

The advantages of this architecture are :

- -it provides a control triggering strategy which is systematic and modular,
- -Rubis model extensions or Meta-base improvements are easily integrated into the Validation Module as a direct result of the flexible representation.
- -the performed controls are independent of what interface is used to input the part of R-Schema, so the Rubis architecture can integrate new interfaces without any modification of the Validation Module.

III.2 Rule Taxonomy

We distinguish four rule classes :

- conformance rules,
- consistency rules,
- completeness rules,
- accuracy rules.

This taxonomy is similar to that found in TODOS [PER.88].

Conformance rules : perform the "syntactic" checking of the R-Schema. We means by this term the syntax of the model and the specification language.

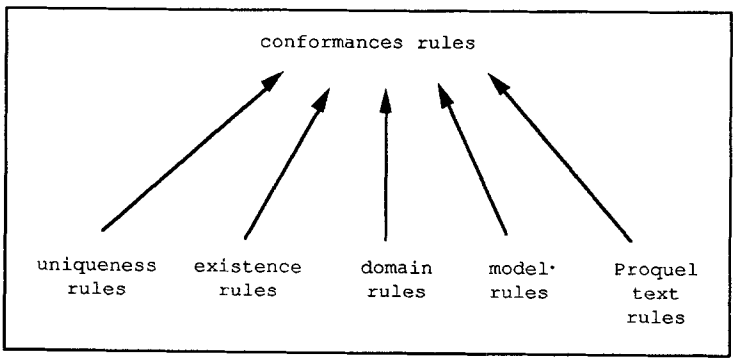

Figure 9 : Hierarchy of conformance rules

In this class, we can see five sub-classes shown by the previous figure.

Uniqueness rules : verify the uniqueness of a schema component. For instance, **"the event code must be unique in the set of event codes"** or **"the attribute name of an object must be unique in the set of object attribute names"**.

Existence rules : check that each R-Schema component is defined. For instance, **"the name of the object which is modified by the operation must be defined in the operation specification"**.

Domain rules : check the value of a schema component according to the domain definition. For instance, **"An event type is either 'internal event' or 'external event' or 'temporal event'"** and **"an operation type is either 'INS' or 'UPD' or 'DEL'"**.

model rules : correspond to cardinality rules between components of the R-Schema. For instance, **"an event must trigger at least one operation"** or **"an operation must modify at most one object"**.

Proquel text rules : express that for each component of the R-Schema expressing Proquel text, must be correct according to the syntax of the Proquel language. In other words that means these texts must be validate by the Proquel Interpreter. For instance, **"an event predicate must be correct according to the Proquel syntax"**.

Consistency rules : check that there is no contradiction in the specifications and that no contradiction can be deduced from the R-Schema.

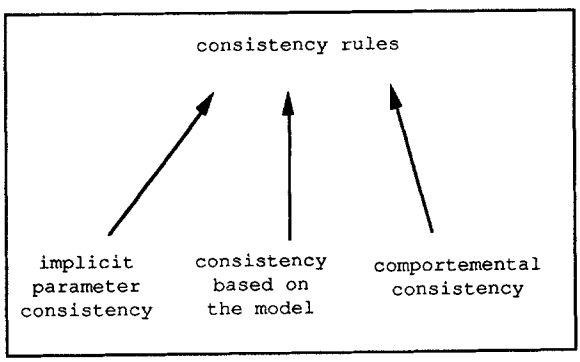

Figure 10 : Hierarchy of the consistency rules

We can decompose this class as the hierarchy illustrated by the following figure.

We present three examples of consistency rules.

The first is associated to the notion of "CONTEXT" in Proquel. We use the implicit parameter "CONTEXT" to refer to the object instance on which an event can be ascertained. The following control can be expressed : **"each field prefixed by "CONTEXT" and used in the operation text must correspond to an attribute of the object on which the event triggering operation is ascertained"**. This rule belongs to "implicit parameter consistency" sub-class.

The second example can be seen, for instance, when an object is accessed with modify statements in condition text, whereas **"a condition text should not modify an object's state"**. This control checks the consistency between the content of Proquel condition text and the condition definition in the model. It belongs to the sub-class "consistency based on the model".

The last case is deduced from the specification when we can have contradictory behaviours of the application in the same specification. This defines the "comportemental consistency" sub-class. As an example of this sub-class, consider the following R-Schema illustrated in figure 11 : in the dynamic transition of the event, an unconditional operation OP1 sets an attribute of OB1 to 2, whereas another unconditional operation OP2 sets the same attribute to 5.

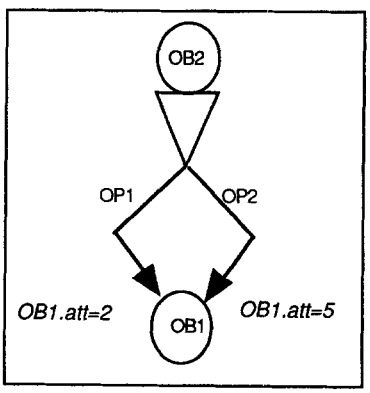

Figure 11 : graphical representation of dynamic transition

Rubis does not recognize execution order between operations belonging to the same dynamic transition. The execution of OP1 before that of OP2, or of OP2 before that of OP1, or their parallel execution must give the same result. It is evident, in this example, when EV1 is fired, the final value of "att" depends on the execution order of OP1 and OP2. The indeterminate result of the dynamic transition is a proof of specification inconsistency. The following rule detects this kind of inconsistency :

R31 : **"When an object instance is modified by more than one operation in the same dynamic transition, the triggering conditions must be mutually exclusive"**.

The detection of these types of inconsistencies can not be automated from the Proquel specification as they are data dependent. This is an interactive rule which can not be implemented using the Proquel language because we want to use a graphical way to explain the situation to the designer.

Completeness rules : verify that there is no isolated or missing component of the R-Schema. We propose the decomposition of this class as follows :

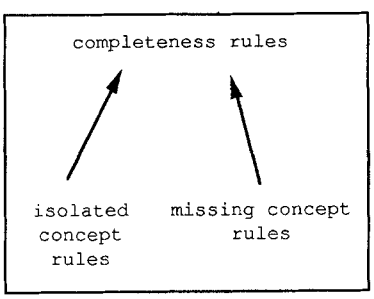

Figure 12 : Hierarchy of the completeness rules

For example, **"an operation must be triggered by at least one event"**. This rule belongs to "isolated concept rule" sub-class.

Another case corresponding to "missing concept rule" sub-class is, **"the name of the object which is modified by an operation and mentioned in the operation specification must be defined as an object with type 'object'"**.

Accuracy rules : detect probable inconsistencies in the R-Schema concerning the accuracy of the specifications as they relate to the application.

These rules point out some critical situations to the designer enabling them to be examined in more detail, thus determining whether or not these situations to the designer are in fact correct.

For example, the vivacity of a dynamic graph is a notion taken from the quasi-vivacity of Petri Net Theory [BRA.83], [BER.79]. This control is expressed by the rule R11 "**each operation that may be triggered by an event may be processed**". We define a *dynamic transaction* as the dynamic transition of an external or temporal event, and with the dynamic transitions of subsequent internal events depending chronologically on the dynamic transition of the preceding event. The rule is checked by presenting each dynamic transaction to the designer and confirming the occurrence feasibility of each internal event and each operation belonging to this dynamic transaction.

Let us consider a second example, the analysis of the *dynamic circuits*. A dynamic circuit is defined by the following rule : "**when an event depends chronologically on itself, we detect a dynamic circuit. It is 'infinite', if the sequence of related events is infinite. This happens when all the operations belonging to this circuit are unconditional and when event predicates are always true after the operation execution. The infinite circuit is not necessarily incorrect but it is forbidden in the Rubis system because it can not be prototyped**". This rule is checked by presenting each dynamic circuit to the designer and its correctness being confirmed.

III.3 Implementation of these rules

We have tried to limit the programming work with the definition of the generic type according to the hierarchy as presented previously. For example, consider the hierarchy of the completeness rules illustrated by figure 12. In our case, the "is-a" hierarchy expresses more the genericity notion issued from abstract data typed languages like ADA [BAR.88] [BOO.88] than inheritance notion coming from object-oriented languages as Smalltalk [PIN.88] [MEY.88].
So, the "Missing Concept" rule and "Isolated Concept" rule are two generic types of rule on which we associate a program model representing the rule. Then, we adapt this program model to the particular situation of R26, for example, by instantiation of input parameters defined in this program model described below.

The program model is a Proquel function as shown in figure 13.

```
FUNCTION missing_concept_rule ($type_mc : STRING,
                               $relation_mc : RELATION_NAME,
                               $attribute_mc : ATTRIBUTE_NAME,
                               $type_wmc : STRING,
                               $relation_wmc : RELATION_NAME,
                               $attribute_wmc : ATTRIBUTE_NAME,
                               $attribute_id_wmc : ATTRIBUTE_NAME)
VAR $result : BOOLEAN;
VAR $x : TUPLE;
BEGIN
FOR EACH $x IN (SELECT [$attribute_wcm], [$attribute_id_wcm]
                FROM [$relation_wcm])
 DO BEGIN
     IF (NOT EXISTS [$relation_mc]
             WHERE [$attribute_mc]=[$x.attribute_wmc])
       THEN BEGIN
             $result:=FALSE;
              affichage_erreur($type_mc,$x.attribute_wmc,
                               $type_wmc,$x.attribute_id_wmc);
            END;
       ELSE $result:=TRUE;
    END;
 RETURN ($result);
END;
```

[$y] : corresponds to the value of the variable $y

Figure 13 : Proquel Function of the "Missing Concept" rules

The following defines the meaning of the input parameters of the program model in figure 13 :

- **$type_mc** represents the type of the 'missing concept',
- **$relation_mc** represents the Meta-base relation (or table) where the 'missing concept' ($type_mc) is stored,
- **$attribute_mc** represents the identifier attribute of the previous relation ($relation_mc),
- **$type_wmc** represents the type of the 'concept' which refers the 'missing concept' ($type_mc),
- **$relation_wmc** represents the Meta-base relation where the previous concept ($type_wmc) is stored,
- **$attribute_wmc** represents the attribute which refers the 'missing concept' ($type_mc) in the previous relation ($relation_wmc),
- **$attribute_id_wmc** represents the identifier attribute of the relation represented by $relation_wmc.

The program corresponding to the rule R26 consists to the following instantiation :

-$*type_mc* : **'operation'** is the missing concept,

-$*relation_mc* : **'ope'** is the Meta-base relation name where is stored all the operations,

-$*attribute_mc* : **'opn'** is the attribute name in the relation 'ope' which represents the operation code,

-$*type_wmc* : **'event'** is the concept where 'operation' is referred and misses,

-$*relation_wmc* : **'trigger'** is the Meta-base relation name where are stored the operations triggered by events,

-$*attribute_wmc* : **'opn'** is the attribute name in the relation 'trigger' which refers the operation triggered by a given event,

-$*attribute_id_wmc* : **'evtn'** is the attribute name in the relation 'trigger' which represents the event code.

We apply this principle for all rules which are automated and we use the Proquel language to implement these rules.

III.4 Control Levels

In this section, we present the definition of three control levels according to three successive development stages of the R-Schema. Then, we describe how we trigger these different levels.

The basic principle which must be respected by the Validation Module is to accept the incomplete specifications because the design process is incremental. Nevertheless, a satisfying level is insured at certain development stages of the R-Schema. We have determined the three following levels :

Level 0 is the imposed level on all the specification interfaces. We have chosen to trigger only a sub-set of conformance rules for keeping the flexibility of each interface given to the designer. For instance, if we define an operation which modifies the object OB2 and OB3, the rule which says that **"an operation modifies only one object"**, is not respected. In this case, the zeroth level of control is not verified.

Level 1 corresponds to the development of the complete conceptual schema where all the executable texts of condition, operation, factor and event predicate are not necessary defined. We check here :

-the conformance rules which do not belong to the previous level and do not concern executable texts,

-the completeness rules which do not concern executable texts,

-the consistency rules which do not need executable texts.

For instance, consider the R-Schema where an operation modifies the object OB1, is defined and this operation is not triggered by any event. This incorrect feature is detected by the rule R26 which expresses that **"an operation must be triggered at least by one event"**. This incompleteness is recognized at this level.

Level 2 corresponds to the development of the executable complete conceptual schema. At this level, all the checking rules based on executable Proquel texts are run, in addition to the rules of the preceding levels. For instance, consider the R-Schema where the object OB1 used in the text of the condition C2, is not defined. This error is detected at this level by the rule R30 which expresses that **"each object used in the text of condition must be defined"**.
Each higher level subsumes the lower levels. The triggering of the different level controls is either automatic or when the designer wants to check the R-Schema.

Level 0 checking is automatically performed when an specification is entered with any design interface. The level 2 checking is either automatically performed when the designer activates the prototyping tools or when the designer wants to check the specification. Level 1 checking is performed when the designer wants to check the R-Schema. Levels 1 & 2 can check the entire R-Schema or just a part of it. The set of controls can be decomposed to the controls on the static or dynamic schemas.

IV Correcting Aid Module

We limit this section to the presentation of the basic principles of the correcting help provided to the designer in the Rubis system and to the brief presentation of the organization of the suggested corrections.

The aim is to assist the designer to correct the errors of the R-Schema highlighted by the checking module.

In general, it is possible to identify two causes of errors, a misunderstanding of the theory underlying and a bad implementation of this theory. The Correcting Aid Module integrates two kinds of help corresponding to two errors classes :

-if a misunderstanding of the Rubis model concepts or Proquel language constructions is the cause of errors, we help the designer by giving the concept definition corresponding to the detected error and providing a set of examples and a set of exercises for correcting the situation. This solution is taken from the "tutoring software" in which the learning strategy is composed of three items: **definitions, examples and exercises** [LEF.84]. We provide this help for each checking rule. For the errors detected by the conformance rules, this help allows to the the designer to correct the R-Schema. This is provided when the designer requests it because of a failure to understand the reasons for the highlighted error.

-if a bad design or bad usage of model concepts, is the cause of errors, we help the designer by the answer to the following question:

What are the changes to be made to correct this situation ?

For each detected error, we suggest to the designer **a set of possible corrections**. The designer can choose one of them or refuse the proposed suggestions. This help is provided when the designer requests it because of the failure to understand how to correct the situation.

For implementing the suggested corrections, we apply the same approach used to implement the checking rules. We use the hierarchy of rules described previously (cf section III.2) and we define for each generic type of rules a set of possible corrections which is implemented by a model program. This model program will adapt to the particular mistakes detected by the checking rules by the different values taken by the input parameters.

We illustrate this approach by the example of the completeness mistakes. The hierarchy of the completeness mistakes is the same as that of completeness rules : "missing component" mistakes and "isolated component" mistakes.

When we have a mistake detected by the "missing component" rule, we can suggest two possibles corrections :

-if the missing component is useful, we define it,

-if the component may cause a referential error by erroneously referring to a non existent component, so we correct the reference.

We can adapt this general situation with the different values of the following input parameters to the mistake detected by a checking rule :

- *tm* : type of the missing component,
- *itm* : instance of the type defined by *tm*,
- *tom* : type of component where tm is referred,
- *itom* : instance of the type defined by *tom*,
- *trel* : type of the relationship between the types *tm* and *tom*.

Let us consider the sub-set of the R-Schema where is defined :

-the objects OB1 and OB3,
-the internal event EV1 occurring on the state change of OB1 and triggering the operation OP1 shown in following figure 14,
-the operation OP1 modifying the object OB2.

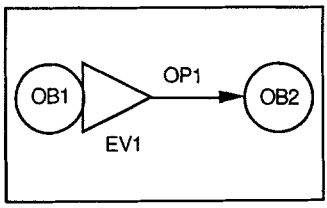

Figure 14 : Dynamic graph of the R-Schema

A mistake is detected by the rule R2 because the object OB2 modified by the operation OP1 is not defined. Each mistake detected by the rule R2 is adapted from the general situation by the following instantiation :

-the type of the missing component (tm) = 'object',
-the type of the component where 'tm' is referred (tom) = 'operation',
-the type of relationship between 'tm' and 'tom' (trel) = 'modified by'.

In our particular mistake, we adapt the help associated to the rule R2 with the following values of the input parameters :

-the instance of the missing object (itm) = 'OB2',
-the instance of the operation where the object is referred (itom) = 'OP1'.

So, we apply this approach for all mistakes detected by the Validation Module.

V Conclusion

In this paper, we have presented the Rubis system which provides :

- -a model and a specification language to aid in the development of the specification of the R-Schema,
- -a module to determine the correctness of the R-Schema,
- -a module to assist in the correction of any mistake detected by the previous module,
- -a prototyping mechanism to allow the execution of the specification on test cases,
- -various interfaces to input the specification in the Rubis system.

We have discussed in more detail the checking rule architecture and the checking rules taxonomy. Then we have presented the principles of the Correcting Aid Module.
These two modules are integrated in the Rubis system implemented in the SUN 3/60 workstation.
The perspective of this work is, on the practical way, to achieve the implementation of these two modules and, on the theorical way to improve the Correcting Aid Module to the architecture of "intelligent tutoring software" [NIC.88] where the module adapt the help to the designer which uses it. Our goal is also to integrate more checking rules in the Validation Module like the quality heuristics for R-Schema improvements or checking rules based on the knowledge of the application domain described in [WOH.88].

ACKNOWLEDGEMENTS : We wish to thank Bob Jansen for reading and commenting this paper.

REFERENCES :

[BAR.88] BARNES J. : "Programmer en ADA" , InterEditions (ed), 1988.

[BER.79] BERTHOMIEU B. " Analyse structurelle des réseaux de PETRI : méthodes et outils ", Thèse de Docteur Ingénieur, Toulouse, 1979.

[BOO.88] BOOCH G. : "Ingéniérie du logiciel avec ADA, InterEditions (ed), 1988.

[BOU.86] BOUFARES F.,ELKABBAT J., JOMIER G.,OUNALLY H. : " Le système de Bases de Données Relationnelles PEPIN3", Rapport de Recherche ISEM N°34, Univ. PARIS SUD, Mai 1986.

[BRA.83] BRAHMS : " Théorie et pratique des réseaux de PETRI", Masson (ed) 1983.

[BRO.82] BRODIE M.L., SILVA E. : "Active and Passive Component Modelling : ACM/PCM", in [CRI.82].

[BRO.83] BRODIE M.L. : "On the development of Data Models", in On Conceptual Modelling, Perspectives from Artificial Intelligence, Databases, and Programming Languages, Edited by Brodie M.L., Mylopoulos J., Smidt J.W., Springer-verlag,1983.

[CAU.88] CAUVET C. : " Un modèle et un outil d'aide à la conception des systèmes d'information ", Thèse de doctorat de l'université Paris VI, 1988.

[CRI.82] " Information Systems Design Methodologies : a comparative Review", Olle T.W., Sol H.G., Verrijn-Stuart A.A. (eds), North-Holland (pub), 1982.

[LEF.84] LEFEVRE J.M. : " Guide pratique de l'E.A.O.", Cedric Nathan, 1984.

[LIN.88a] LINGAT J-Y., COLIGNON P., ROLLAND C. : "Rapid Prototyping : the PROQUEL Language", Proc. of the 14 th VLDB Conference, Los Angeles, 1988.

[LIN.88b] LINGAT J-Y : " RUBIS : un système pour la spécification et le prototypage d'applications Bases de Données ", Thèse de doctorat de l'université Paris VI, 1988.

[MEY.88] MEYER : " Objected-Oriented Software Construction ", Interactive Software Engineering, 1988.

[NIC.88] NICAUD J.F., VIVET M. : "Les tuteurs intelligents : réalisation et tendances de recherche", TSI Vol 7, 1988.

[NOB.88] NOBECOURT P., ROLLAND C., LINGAT J-Y. : " Temporal Management in an Extended Relational system ", BNCOD6. Conference, England, July 1988.

[ROLL.82] ROLLAND C., RICHARD C. : " The REMORA Methodology for Informatiion systems Design and Management " in [CRI.82].

[ROLL.87] ROLLAND C., FOUCAUT O., BENCI G. : " Conception des systèmes d'information : la méthode REMORA ", Eyrolles (ed), 1987.

[ROLL.88] ROLLAND C., CAUVET C., NOBECOURT P., PROIX C., COLIGNON P.,LINGAT J-Y., SOUVEYET C. : " The RUBIS system ", CRIS88, Computerized Assistance during the Information System Life Transition, September 1988.

[SMI.77a] SMITH J.M., SMITH D.C.P. : " Database Abstractions : Aggregation", Comm. of ACM, Vol 10, n°6, 1977.

[SMI.77b] SMITH J.M., SMITH D.C.P. : " Database Abstractions : Aggregation and generalization", ACM Trans. on Database Systems, Vol 2, n°2, 1977.

[WOH.88] WOHED R. : "Diagnosis of conceptual schemas", IFIP WG2.6/ WG8.1 Working Conference on "the role of Artificial Intelligence in Databases and Information Systems", Canton, July 1988.