A PROCESS MODEL FOR DATA BASES

Jacques Guyot
Centre Universitaire d'Informatique
Université de Genève
12, rue du Lac CH-1207 Genève

Abstract: we present a process model for data bases, which integrates concepts of relation, process, integrity constraint, event and period. The model allows one to specify the synchronization of processes and the constraints due to consumption and production of data. In particular we present the formalization which is carried out with the Petri Nets. These are obtained by using refining primitives which keep the interpretation simple for designers. The aspects of verification and validation of the specification are also examined.

keywords: Process, Data Base, Design, Verification, Petri Net.

1. Introduction

At present the designer is well armed to tackle the specification of data. What he lacks, however, are conceptual tools to :

- specify the processes of an application
- study the interactions between the processes and the data
- define the synchronization constraints between the processes.

The dynamic approach (in comparison with one that only takes the data into account) enables the designer to specify the constraints of utilization belonging to the field of application being considered. The designer can thus answer the following questions:

- What is the availability interval of each data?
- Which data is produced and consumed by the processes?
- What are the synchronization demands between processes? Might the latter cause deadlock situations?
- What are the periods when it is relevant to validate the integrity rules and allow the processes to run?

To answer these questions, the data models (relational, entity-relationship, ...) as wide as they may be, are not powerful enough. For example, two administrations could have an identical data specification, with totally different constraints because the rules and utilizations of the information are specific to each administration. Besides, the study of the above questions has to include an idea of time.

The studies in this field have developed in different directions. The DADES [OLI82] methodology analyzes the data-flow; each data is associated with a avaibility interval and derivation rules describing the data produced and consumed by the processes. This method allows the analysis of the data derivability with regard to time and processes. IDA [BOD83] describes an information system model in terms of events, processes and resources. Using quantitative information on the model's objects, the behaviour of the information system can be studied by simulation before it is refined and implemented. Other conception methodologies REMORA [ROL82], USE [WAS82], MERISE [TAR83] have also included dynamic aspects, although time is not the central point of these studies.

We present MTG [GUY86] which integrates notions of time, processes synchronization and data-flow. In order to specify and verify the different synchronization constraints, this model uses the Petri Net (PN).

We can summarize the five model's concepts as follows:

- The **relations** define the application's objects which have to be stored in the D.B.
- The **integrity rules** express the property governing the applications objects.
- The **processes** are associated to the change of the objects value.
- The **events** describe the synchronization conditions of the processes.
- The **periods** define the working context of the DB; each of which is characterized by its own set of integrity rules and events.

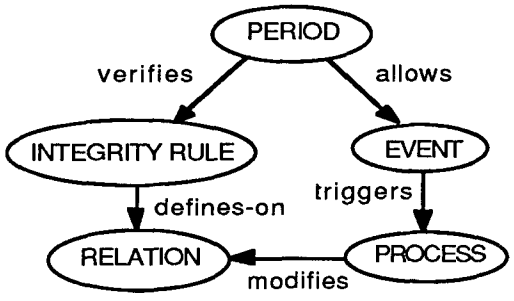


Figure 1 : The MTG concepts and their interactions.

In order to allow the formal verification of the specification, we have chosen the Petri Net which is well adapted to the problems of synchronization and to the resources management. The difficulties inherent in the designing of events synchronization are surmounted by the use of refining primitives (sequence, parallelism, alternative). These allow a modular and top-down design of synchronization. Each of these primitives is associated to a refining operation acting at the level of PN. An algorithm based on refining primitives can be used by the designer. This assures the liveness of events specification (which implies the absence of deadlock). Parts 2 and 3 of this paper will present these aspects.

In part 4, we will develop the consumption/production of data by the processes which is expressed by means of data-flow. Taken from processes specification we describe it in terms of PN. Once the completeness and coherence of data-flow has been verified the designer can be assured that:

- The data of DB can be derived from existing processes and parameters.
- The synchronization constraints between periods are compatible with the data-flow constraints.
- The synchronization constraints on events during a period are compatible with those of the data-flow

Finally, we will briefly discuss the integration of these specifications in a processes dictionary.

2. The DB design and the Petri Net

In [PET80] C. A. Petri set four fields where the PN can be applied:

- the interconnection between various conceptual levels;
- the concurrence (partial independence of occurrences);
- the limitation of resources;
- the research of the most relevant concept at each level.

The PN appears well suited for the study of dynamic behaviour of data base application. Indeed, the occurrence of events is partially independent, the consumption/production data by processes can be transposed as a problem of resources management. The specification of different concepts, using PN, allow the interconnection of them for verification.

The PN has already been used in the specification of DB. For example: in the study of concurrency between transactions [LEO81], in a condition/action model [DEA81], in the specification of an information system [RIC81] [ELI80], and furthermore, in the MERISE methodology [TAR83]. The criticism usually expressed is "*These tools are too formal and are thus reserved to a restricted category of specialists*" [BOS86]. The intensive use of PN as a specification tool must bypass two difficulties expressed by the critics. One concerns the weak abstraction level of the elements constituting the PN (places, transitions, input/output function, tokens) which, if they are used directly, lead to complex PN. The other difficulty deals with the verification carried out on the specification which, for PN made up of many elements, leads to a combinatorial explosion of cases to be examined (for example, the method using the reachability tree [GIR83]). It is thus the expression and the verification of the specification which must be facilitated.

To reach this aim, we structure the specification with the help of refining primitives. For example: sequence, parallelism and alternative which are elements of synchronization that the designer is familiar with. The latter expresses himself in a language which has no relation to the classical representation of the PN. On the other hand, each refining primitives possesses a PN translation which preserves the formal aspect of the specification. The verification is simplified because the refining primitives have properties that authorize a structural approach which facilitates the analysis of the PN [BRA82].

The refining primitives and structural analysis thus allow the designer to express himself in a high level language and to use the PN as a low level representation on which he can verify some properties. This approach is comparable to those of [LAU75] [QUE81] in the field of real time applications.

Petri Net Notations

A Petri Net structure, N, is the triple N=<P,T; F> where

P is a finite set of places
T is a finite set of transitions
F is the input/output function

and

1) $P \cup T \neq \emptyset$
2) $P \cap T = \emptyset$
3) $F \subseteq (P \times T) \cup (T \times P)$

$.x = \{ y \mid (y,x) \in F\}$ is the input set of x

$x. = \{ y \mid (x,y) \in F\}$ is the output set of x

A marked PN is a pair <N,M> where

1) N is a PN

2) M is a function from P to the nonnegative integers

M(p) define the number of tokens in the place p

2.1. The refining primitives

These correspond to the different situations encountered during the specification of events. This specification is associated to a PN in the following way. The occurrence conditions of events correspond to the places and the events are represented by the transitions. Two types of refinement are put in evidence. Firstly, the chronological refinements which allow the designer to replace one event by others having the same effect on the DB but being ordered by a temporal constraint.Those are **sequence**, **parallelism** and **alternative**. The second category groups the synchronization refinements and enables the expression of constraints relative to the occurrence conditions of events. Those are the **simultaneity** of two events, the order of **precedence** between two events and the **equivalence** of two conditions. Moreover, the designer is given a primitive named **embryo** which corresponds to an initial activity belonging to the application.The details of this activity will be refined by the use of refining primitives. Another primitive named **union** is used to put together distinct specifications of events in order to express the synchronization constraints affecting them. Finally, two technical primitives (RedCond and RedEvent) can be used to eliminate the conditions and the events which have not been used during the specification.

In the following sections, we show the refining primitives under two different angles. The first is related to the designer and his interpretation of primitives during the specification of the field of application. The second is in regard to transformations that the refined PN undergoes (here we only give a graphic representation of the transformations, see [GUY86] for a formal definition).

Sequence: SEQ(N, e, E_S, C_S)

The refining primitive sequence applies each time that the designer has to detail an event **e** in the specification **N** by replacing a set of events $\mathbf{E_S}$. The events of E_S have the same consequences as the replaced event. The sequential order of the events E_S is imposed by the set of conditions $\mathbf{C_S}$.

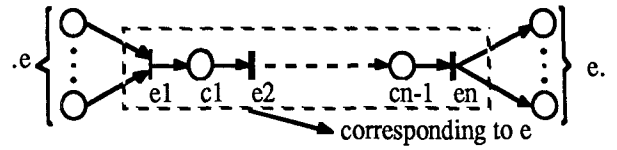


figure 2 : Transformation of Petri Net, N, for the primitive SEQ(N, e, E_S, C_S)

Parallelism: PAR(N, e, E_p, C_p)

This refining primitive is used to replace an event e by a set of events $\mathbf{E_p}$ which have the same consequences as e. Moreover, no other constraint exists between the events of E_p. They are thus independent one from another.

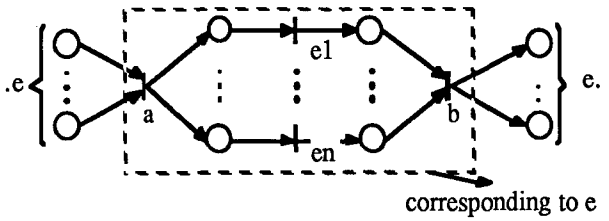


figure 3 : Transformation of Petri Net for the primitive PAR(N, e, E_p, C_p)

Alternative: ALT(N, e, E_a)

This primitive allows the designer to replace an event **e** by a choice of events $\mathbf{E_a}$ each having the same consequences as the replaced event (the choice can depend on data context)

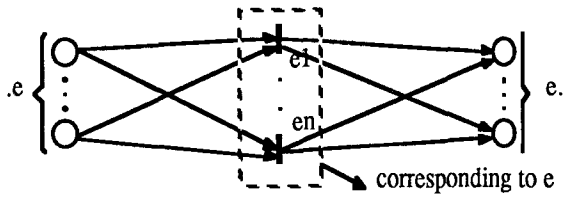


figure 4 : Transformation of Petri Net for the primitive ALT(N, e, E_a)

Simultaneity: SIM(N, e_S, e_1, e_2)

This primitive expresses the fact that two events $\mathbf{e_1}$ and $\mathbf{e_2}$ always happen at the same time. The designer can use it in two different cases. Either he expresses a synchronization constraint relating to two distinct events or he wants to unify two specifications designed separately and he uses simultaneity to take the common events into consideration.

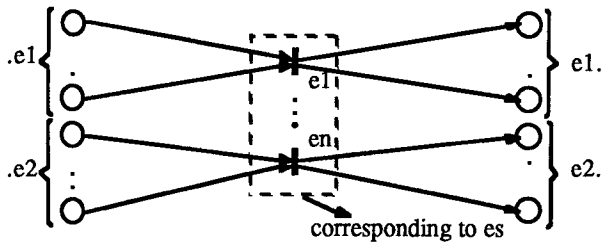


figure5: Transformation of Petri Net for the primitive SIM(N, e_s, e_1, e_2)

Precedence: PRED(N, e_1, e_2, c_p)

The designer uses this primitive so as to express the order of precedence between two events e_1 and e_2. This means that e_1 occurs before e_2

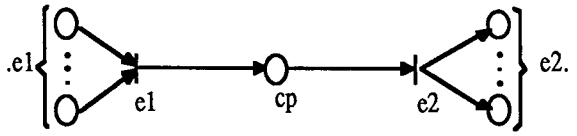


figure 6: Transformation of Petri Net for the primitive PRED(N, e_1, e_2, c_p)
(the place c_p is added to the PN)

Equivalence: EQU(N, c_e, c_1, c_2)

It allows the designer to merge two conditions c_1 and c_2 into one. As in the case of simultaneity, two cases are possible. The one represents the conjunction of two distinctive conditions and the other represents the unification of two identical conditions belonging to different specifications.

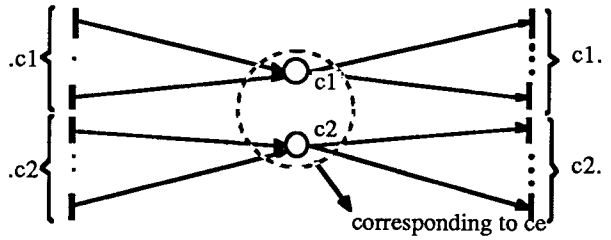


figure 7 : Transformation of Petri Net for the primitive EQU(N, c_e, c_1, c_2)

Embryo: EMB(c_{emb},e_{emb})

The embryo corresponds to an activity in the field of application and is the starting point of all the refinements of this activity.

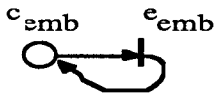


figure 8 : PN corresponding to the embryo: EMB(c_{emb},e_{emb})

2.2. Example of specification

Here are the terms of the problem. There are two activities, one of production and one of consumption. A consumer asks for an estimate for the fabrication of an object. The producer manufactures or assembles the elements with the view to satisfying his customer. The planning of the fabrication is only undertaken once the customer has paid a deposit. After fabrication, tests are made on a sample and at the same time the objects are packed. Then the finished product is delivered and the customer pays the balance.

We will proceed in three steps:

- describe the activity of consumption;
- describe the activity of production;
- find the constraints binding the two activities.

specification of the production

1) We start with an embryo.
2) We carry out a sequential refinement (establish_estimate -> signature_contract -> planning -> realization -> prepare -> expedition -> delivery).
3) The realization of an object can be obtained by manufacturing or by assembly.
4) The preparation of expedition needs the testing of a sample and the packing of the rest of the objects. These operations are carried out at the same time.

specification of the consumption

1) We start with an embryo.
2) We carry out a sequential refinement (request_estimate -> signature_estimate -> payment_deposit -> payment_balance).

specification of constraints

1) The union of two specifications needs to be executed.
2) The signature of the contract by the producer and the consumer is simultaneous.
3) The request of an estimate precedes its establishment.
4) The payment of a deposit precedes the planning.
5) The delivery precedes balance payment.

By using the specification language described in the appendix, the three steps are described as follows:

```
SPECIFICATION
example1 <-
   UNION( EMB(c1,activity_production),
          EMB(c1,activity_consumption))
   REFINED_BY
      SEQ( activity_production,
           {establish_estimate,
           signature_contract,
           planning,
           realization,
           prepare_expedition,
           delivery},
           {c2,c3,c4,c5,c10});
      PAR( prepare_expedition,
           {testing,packing},
           {c6,c7,c8,c9});
      SEQ( activity_consumption,
           {request_estimate,
           signature_estimate,
           payment_deposit,
           payment_balance},
           {c12,13,14});
      SIM( acceptation,
           signature_contract,
           signature_estimate);
      PRED(request_estimate,
           establish_estimate,c15);
      PRED(payment_deposit,planning, c16);
      PRED(delivery,payment_balance,c17);
      ALT( realization,
           {assembly,manufacturing});
END
```

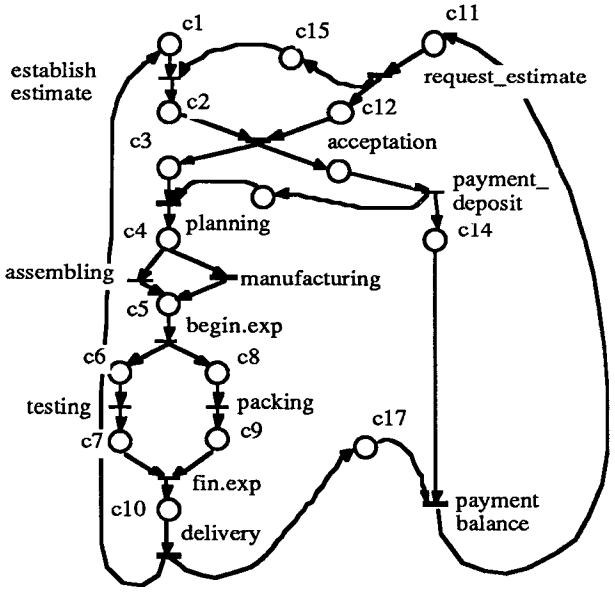


figure 9 : PN corresponding to the specification of the example1

The PN obtained by the different refinements is described in figure 9. We have thus obtained a specification of events which allows two representations. One is accessible to the designer and the other, that of the PN, allows a formal verification of the synchronization properties by the specification.

3. Verification of events specification

When dealing with synchronization, one of the essential points is to verify the absence of deadlock. This verification is carried out directly at the level of the PN by analysis of *liveness*. In our case, the PN is obtained by the application of refining primitives which thus lead to the study of primitives related to liveness. The first result concerns the power of expression of refining primitives.

Property A

For all PN, N=(P,T;F), there exist a sequence of refinements using the following primitives: embryo, union, precedence, equivalence, reduction of place or reduction of transition, which builds N.

This lead us to the study of the PN in general. But in this case, we know that the complexity of the verification of liveness is NP-complete [PET81]. However, the refining primitives possess good properties in relation to liveness which can be exploited if they are used in a correct order. The proposed algorithm reflects this order.

Property B

Let N be a PN and N' be the result of the application on N of one of the following primitives: sequence, parallelism, alternative.
Then we have a live <N',M'> if and only if we have a live <N,M>
where $M'(p) = M(p)$ if $p \in P$
$= 0$ otherwise

This property implies that the following primitives: sequence, parallelism and alternative conserve the liveness of the refined PN. Since we choose the PN embryo as starting point it is sufficient to note that if its only place is marked we have a live PN.

The property C concerns simultaneity and precedence primitives. It is restricted to Marked Graph (MG) which constitutes a sub-class of PN. It indicates that it is sufficient to verify that all the paths which lead to the two transitions (involved in the refinement) are marked, i.e. that a place of the path at least possesses a token.

Property C

Let <N,M> be a live MG , N=<P,T;F> and $t_1, t_2, \in P$

a) Let N' <- PRED (N,t_1,t_2,c), c $\in$ P'
if all the path from t_2 to t_1 are marked then we have a live <N',M'>
where M'(p) = M(p) if p $\in$ P
= 0 otherwise

b) Let N' <- SIM (N,t_1,t_2,c), c $\in$ P'
if all the path from t_2 to t_1 and from t_1 to t_2 are marked then we have a live <N',M'>
where M'(p) = M(p) if p $\in$ P
= 0 otherwise

We also know that sequence and parallelism preserve the adherence to the sub-class of Marked Graph. This lead us to establish the algorithm A of specification and to put into evidence the property D.

Algorithm A

a) creating the embryos N_i=EMB(c_i,e_i) with the marked placed c_i, i=1..n
b) unifying embryos N=UNION(N_1, ..., N_n)
c) using the refining primitives
- sequence
- parallelism
- simultaneity while verifying the condition of the property C
- precedence while verifying the condition of the property C

d) using the refining primitives
- sequence
- parallelism
- alternative

Property D

Each specification obtained by following the algorithm A gives a live PN.

In the previous example we can verify that it follows the rules stated in the algorithm, thus we have a live PN.

This algorithm does not construct all the alive PN that exists. However, it covers the designer's needs in most cases. By using this algorithm, the designer is obliged to structure the specification in a top-down way. In addition, the algorithm assures him that his specification of events is free of deadlock.

4. The data-flow

The specification of events does not take into account the use of the data. This is described by the specification of the processes. For each process T_i, we can describe the following sets:

I_i: set of relations (or views built on existing data in the DB) consumed by the process
Par_i: set of information belonging to the field of application (given by the user and not memorized in the DB)
O_i: set of relations (or views build on existing data in the DB) produced by the process

If TRAIT={T1, ..., Tn} is the set of processes of the application then

PAR = $\cup_{i=1,n} PAR_i$ is the set of parameters of the application

I = $\cup_{i=1,n} I_i$ is the set of input of the application

O = $\cup_{i=1,n} O_i$ is the set of output of the application

As we have already underlined, our aim is to use a unique formalism, that of the PN. For a process T_i we obtain the following transcription: the elements of PAR_i and of I_i are input places of the transition T_i (the process) and the elements of O_i the output places of T_i

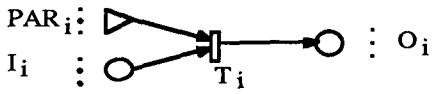


figure10 : PN corresponding to the process T_i

By extending this to the set of processes, we obtain:

Graph of data-flow

Let TRAIT be a process specification. A data-flow G of TRAIT is the triple G = <P,T;F>
where P = I $\cup$ PAR $\cup$ O, the set of information
T = TRAIT = {T_1, ..., T_n}
F = { (p,T_i) | $T_i \in$ T et p $\in PAR_i \cup I_i$}
$\cup$ { (T_i,p) | $T_i \in$ T et p $\in O_i$}

The input/output function of the PN formalizes the notion of consumption/production of the process information. From this we can define the desired properties of specification. For example: the well-formed specification and the derivability of an information.

Well-formed specification

A specification of processes TRAIT is well-formed if and only if

1) $PAR \cap I = \emptyset$
2) $PAR \cap O = \emptyset$
3) $I \subset O$

These three constraints are necessary to avoid any ambiguity in the description of the data. 1) and 2) express that the parameters are external to the DB. 3) shows that all the input data of a process comes from the DB and that it is thus produced by another process.

Derivability

Let G=<P, T; F> be the data flow of a well-formed specification TRAIT.
$p \in P$ is derivable if and only if

1) $\exists\ t \in T$, where $p \in t.$ and $\forall\ x \in .t$, x is derivable

or 2) $.p = \emptyset$ (source place)

By extension, G is derivable if and only if
$\forall\ p \in P$, p is derivable.

For the designer, the derivability of a processes specification means that for all the data there exists at least one sequence of processes which, taken from the parameters, produces the desired information. The **derivation sequence** of an information is a sequence of processes to be executed to produce this information.

4.1. The compatibility between event specification and process specification

If we go back to figure 1 which represents all the concepts, we can notice that event and process specification are not independent. The occurrence of an event triggers off the execution of a process. This implies that the designer has to verify the compatibility between synchronization constraints and constraints imposed by the data-flow.

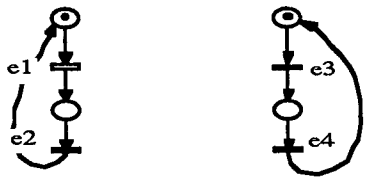
To verify the compatibility between these two types of constraints, we transform the derivation sequences (which produce output data) into constraints of precedence applied on the specification of events. And, once again, the designer verifies that the specification stays free of all deadlock (this can be done with the property C)

Let us illustrate this with a simple example:

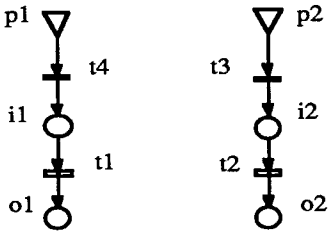
We have the following specification for events:

```
SPECIFICATION
example2 <- UNION(EMB(c1,e10),EMB(c3,e20))
   REFINED_BY
      SEQ(e10,{e1,e2},{c2});
      SEQ(e20,{e3,e4},{c4});
END
```

Thus we have the following PN:



Let TRAIT be a process specification having the following data-flow:



Moreover, we have the following trigger function:

trigger (e1) = t1
trigger (e2) = t2
trigger (e3) = t3
trigger (e4) = t4

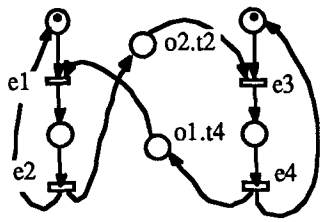
The derivation sequences of o1 and o2 are:

Derivation (o1) = { <t4,t1>}
Derivation (o2) = { <t3,t2>}

This means that t4 must precede t1 and t1 must precede t3. Thus, we add the following constraints of precedence to the event specification:

PRED(example2, t4, t1, o2.t2)
PRED(example2, t3, t2, o1.t4)

We obtain the next non-live PN (a non-marked path is encountered when we add the second constraint)



Thus by introducing the precedence constraints one by one, it is possible to know the data of which the derivation is problematic. The designer can thus analyze the existing incompatibilities.

5. Integration of specification in a dictionary

This last section deals with a specification dictionary. In the proposed model, each concept is specified independently from the others. This has the advantage of making a good separation of the functionality of the concepts:

- The processes define the transformation of data.
- The relations define the conceptual schema of the DB.
- The events define the synchronization and the triggering of the processes
- The integrity rules define the coherent states of the DB.
- The periods define the authorized events and the integrity rules to be verified.

This aspect of the model allows the postponement of implementation decisions until the moment of its realization. For example, it is at the last moment that we will decide to introduce an integrity rule in a process to be validated.

To manage this independence, we have specified and implemented a dictionary which accepts the specification of an application. Each concept is specified in its own language. The specification texts are then compiled. When the five parts are memorized in the dictionary, it is possible to carry out an analysis of the completeness and coherence of the specification.

In the next step, we validate the specification by executing it. The specification of the periods and the events is activated by a automaton which simulates their behaviour. The processes are interpreted, they read and modify the data in a DB defined by the conceptual schema. The execution is carried out under the control and the remarks of the final user of the system. This allows the specification to be corrected as quickly as possible. The execution of the specification is carried out directly on the data of the dictionary, thus avoiding any introduction of errors (refer to [GUY85] for further details).

Any eventual corrections are supported by the interrogation of the dictionary which allows the binding of the different concepts. We can know for example:

- the list of relations, attributes, keys, ..;
- the periods which authorize the modification of a given relation;
- the processes which modify a given relation;
- the events directly anterior or posterior to a given event;
- the processes that can invalidate a given integrity rule;
- etc.

6. Conclusions

We have presented a model of processes for DB. It allows the formalization of constraints specific to the processes such as the synchronisation of events or the data-flow. The Petri Net is used to formalize these constraints and to analyze the dynamic behaviour. However, to render this utilization operational for the designer, the refining primitives are introduced. They allow a top-down and structured design. An algorithm enables the designer to detect and eliminate the deadlocks in the event specification. The use of a unique formalism allows the verification of compatibility of various aspects formalized with the PN.

Finally, we describe a process dictionary which allows the memorization of the specification and to obtain an immediate execution from it. This facilitates the validation of the application by the user.

Our research is now orientated towards the following steps of design, that which consists of passing from the specification to an operational version of application. In the first stage, we wish to use all information found in the dictionary so as to guide the designer in the choice of an implementation.

BIBLIOGRAPHY

[BOD83] Bodart, F; Pigneur, Y.
"Conception assistée des applications informatiques 1. Etude d'opportunité et analyse conceptuelle." Masson, 1983

[BOS86] Bosc, P.; Bouzeghoub, M.; Chrisment, C.; Flory, A.; Jomier, G.; Miranda, S.; Rolland C.; Spaccapietra, S.
"Interfaces bases de données avancées: quelles recherches pour quelles interfaces?". Journal Modèles et Base de Données, N4, sept. 1986.

[BRA82] Brams, C.W.
"Réseaux de Petri: théorie et pratique". Edition Masson, 1982.

[DEA81] De Antonellis, V.; Zonta, B.
"Modelling Events in data base application design". Proc 7th VLDB, Cannes, 1981.

[ELI80] Ellis, C.; Nutt, G.
"office Information Systems and Computer Science". ACM Computing Surveys, V12 N1, March 1980.

[GIR83] Girel, B.
"Outil pour la manipulation et la représentation graphique des Réseaux de Pétri" Mémoire de diplôme, Cahier du CUI No35, Université de Genève.

[GUY85] Guyot, J.
"La spécification des traitements dans les bases de données et son exécution: un prototype" Actes INFORSID Luchon, 1985.

[GUY86] Guyot, J.
"Un modèle de traitement pour les bases de données: un formalisme pour la spécification, la validation et l'exécution de la spécification d'une application". Thèse de la Faculté des Sciences, Concept Moderne/Editions , Genève, Sept 1986.

[LAU75] Lauer, P.E.; Campbell, R.H.
"Formal semantic of a class of high-level primitives for coodinating concurrent processes" Acta Informatica 5, 1975.

[LEO81] Léonard, M.; Luong, B.
"Information System Design Approach Integrating Data and Transactions". Proc. 7th VLDB, Cannes, 1981.

[OLI82] Olivé, A.
"DADES: a methodology for specification and design of information systems." Information system design methodologies: a comparative review.North Holland, 1982.

[PET80] Petri, C.A.
"Introduction to general net theory". in Net theory and application, Springer Verlag, 1980.

[PET81] Peterson, J.L.
"Petri net theory and the modeling of systems" Prentice Hall, 1981.

[QUE81] Queille, J.-P.
"The CESAR system: an aided design and certification system for distributed applications". 2th conference of distributed computing system IEEE, Paris, 1981.

[RIC81] Richter, G.
"IML - inscribed nets for modeling text processing and data base management". Proc. 7th VLDB, Cannes, 1981.

[ROL82] Rolland, C.
"The REMORA methodology for information system design and management" Information system design methodologies: a comparative review.North Holland, 1982.

[TAR83] Tardieu, H.; Rochfeld, A.; Colletti, R.
"La méthode MERISE: Principles et outils". Les éditions d'organisation, 1983.

[WAS82] Wasserman, A.
"The user software engineering methodology: an overview" Information system design methodologies: a comparative review.North Holland, 1982.

APPENDIX: Language of refinement

Notations:
- in bold; terminal symbols
- in plain; non-terminal symbols
- | ; alternative
- [] ; optional expression
- {} ; repeat 0 or more an expression

specifation := **specification**
assign {assign}
end

assign := PN_ident <- pn
[**refined_by** primitive ; { , primitive ;}]

pn := PN_ident
| **emb** (cond_ident , event_ident)
| **union** (pn { , pn})

primitive :=
seq (event_ident , list_event , list_cond)
| **par** (event_ident , list_event , list_cond)
| **alt** (event_ident , list_event)
| **pred** (event_ident , event_ident , cond_ident)
| **sim** (event_ident , event_ident , event_ident)
| **equ** (cond_ident , cond_ident , cond_ident)
| **redcond** (cond_ident)
| **redevent** (event_ident)

list_event := { event_ident { , event_ident } }

list_cond :={ cond_ident { , cond_ident } }

PN_ident = event_ident = cond_ident = string of chararcters.